Introduction to Clustering

Ludwig Krippahl

Introduction to Clustering

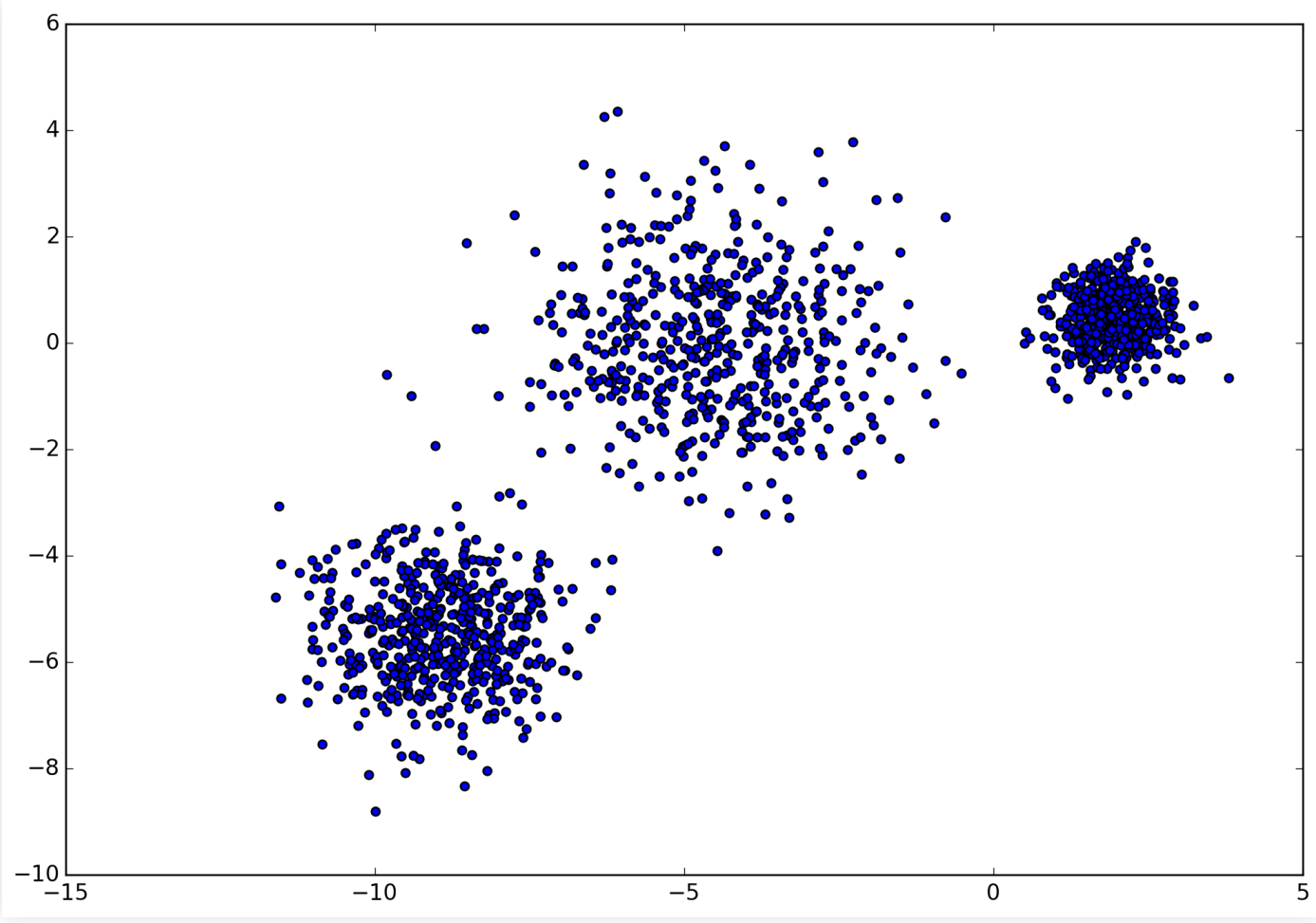
Summary

- Introduction to clustering
- K-means and k-medoids
- A first (informal) look at Expectation-Maximization

Clustering

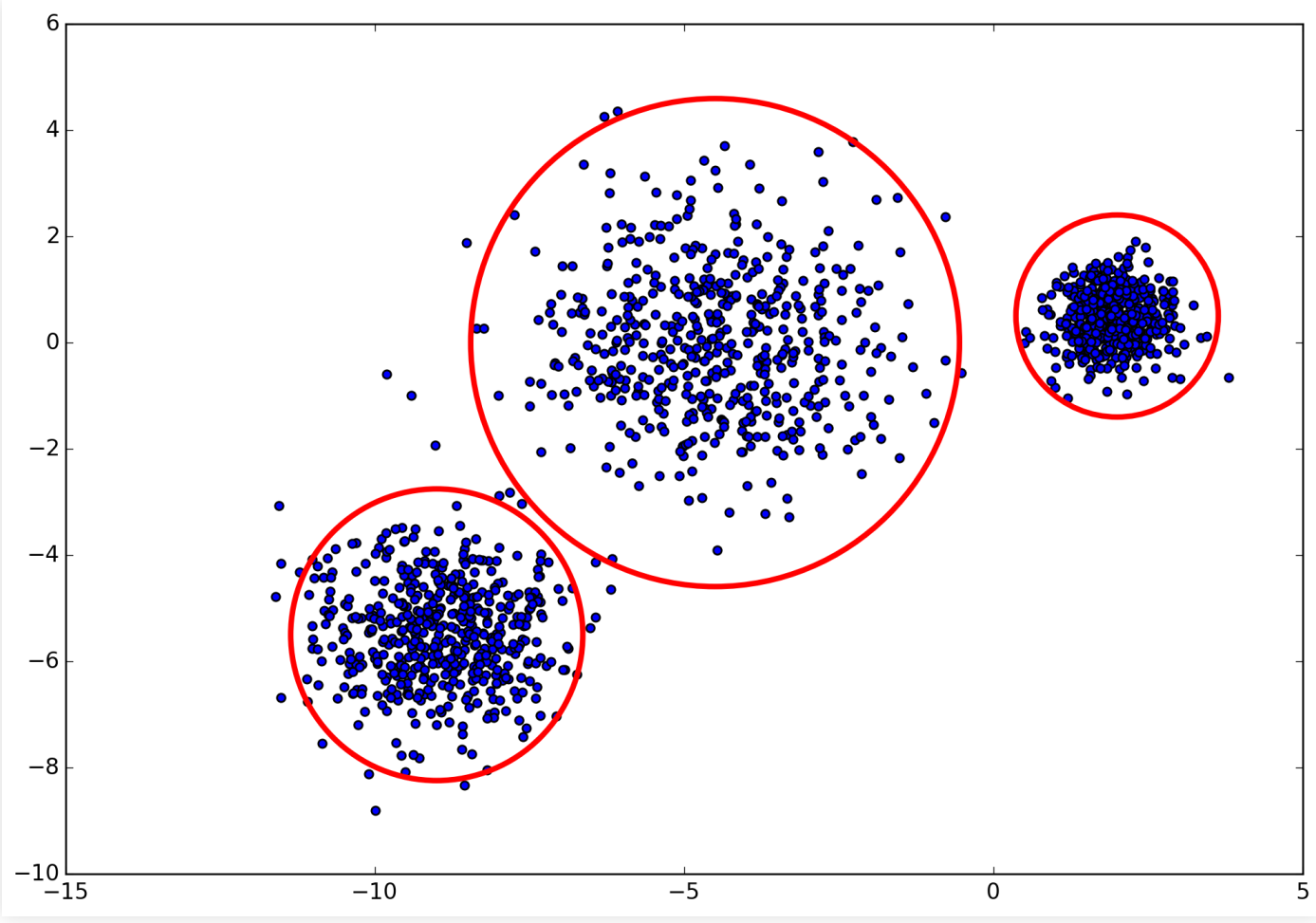
Clustering

Basic idea: Group similar examples together



Clustering

Basic idea: Group similar examples together



Clustering

Basic idea: Group similar examples together

- Min difference (or max similarity) within clusters
- Max difference (or min similarity) between clusters
- Clustering requires some measure of similarity or difference

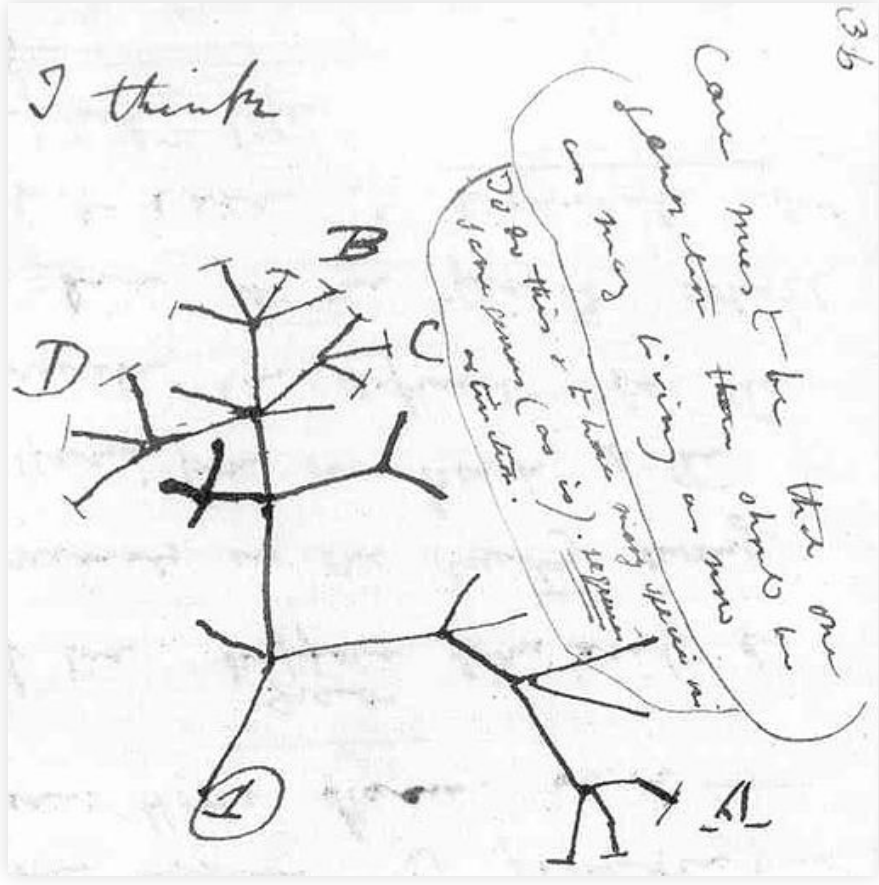
Why clustering?

- To help understand data and relations

- Meaningful grouping is part of how we understand things
- Clustering helps make sense of complex or abundant data
- WWW searches, biology, climate, ...
- Clustering can help identify meaningful groups

- To reduce data to relevant examples

Darwin's "tree of life"



Source: Wikimedia Commons, public domain.

Identifying "Ghost Cities"

"Ghost Cities" Analysis [...]. Chi et. al, http://arxiv.org/abs/1510.08505

Why clustering?

- To reduce (summarize) data:

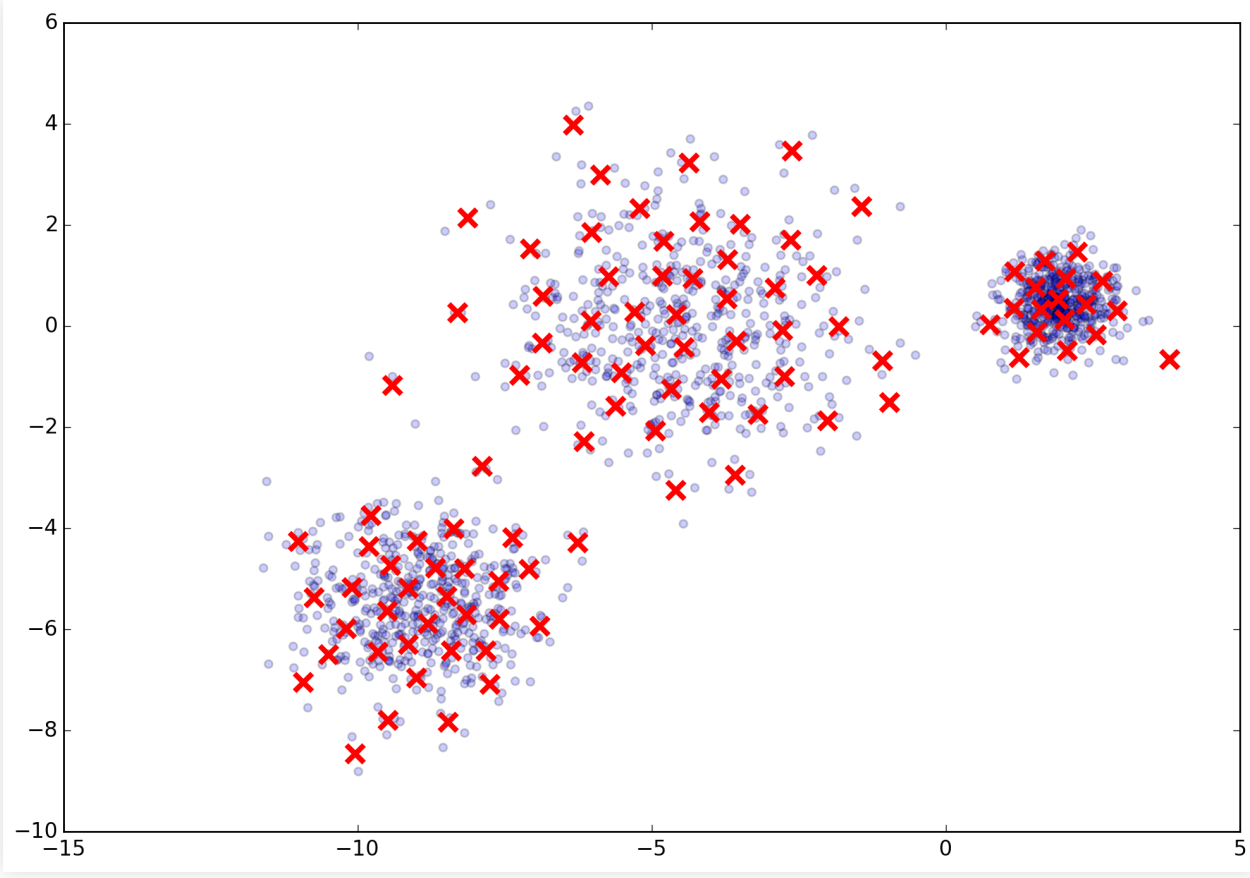
- Cluster prototypes abstract properties from groups and can be used to reduce the data set replacing all points in group

- Can improve performance of data analysis or classification (often quadratic on the number of examples)

- Sumarizing, compressing, finding neighbours, image segmentation, etc

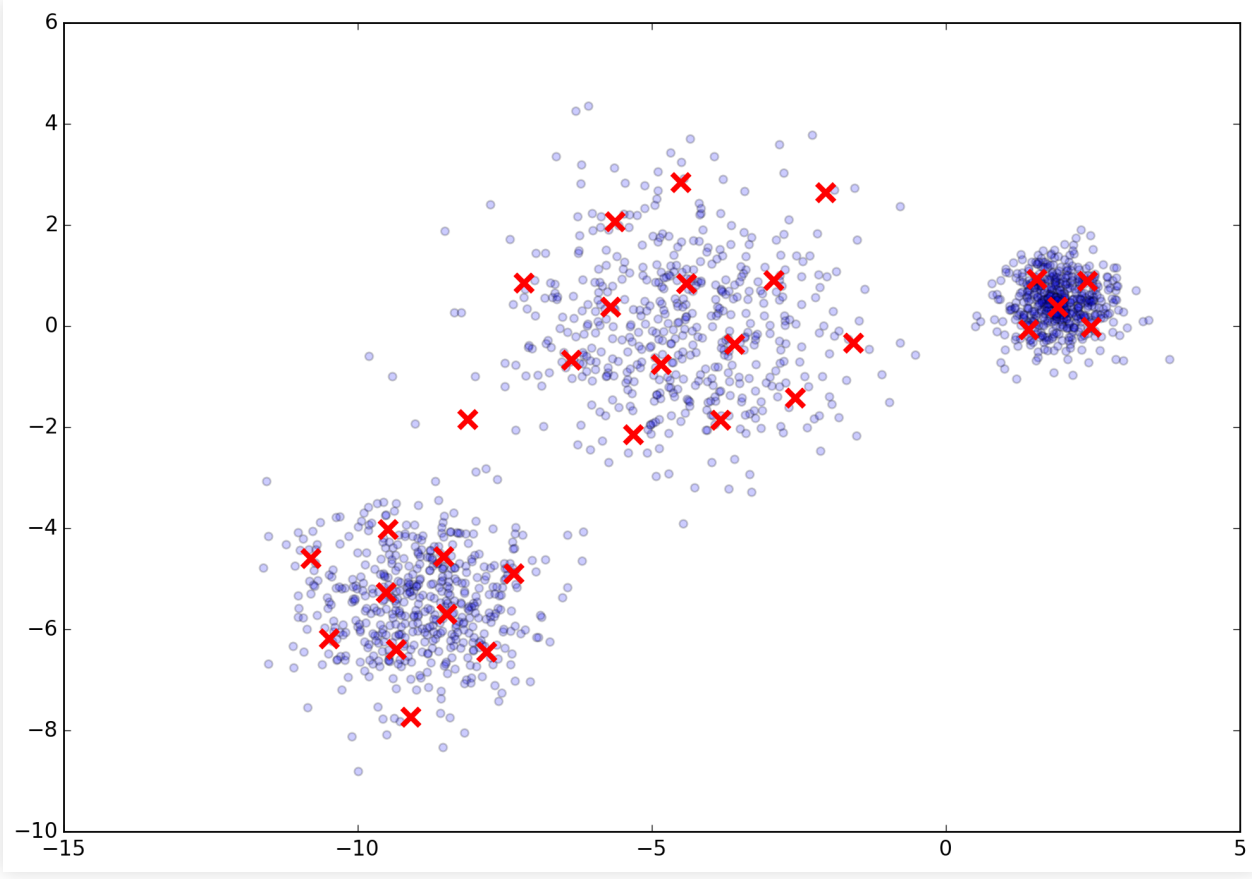
Why clustering?

To reduce (summarize) data:



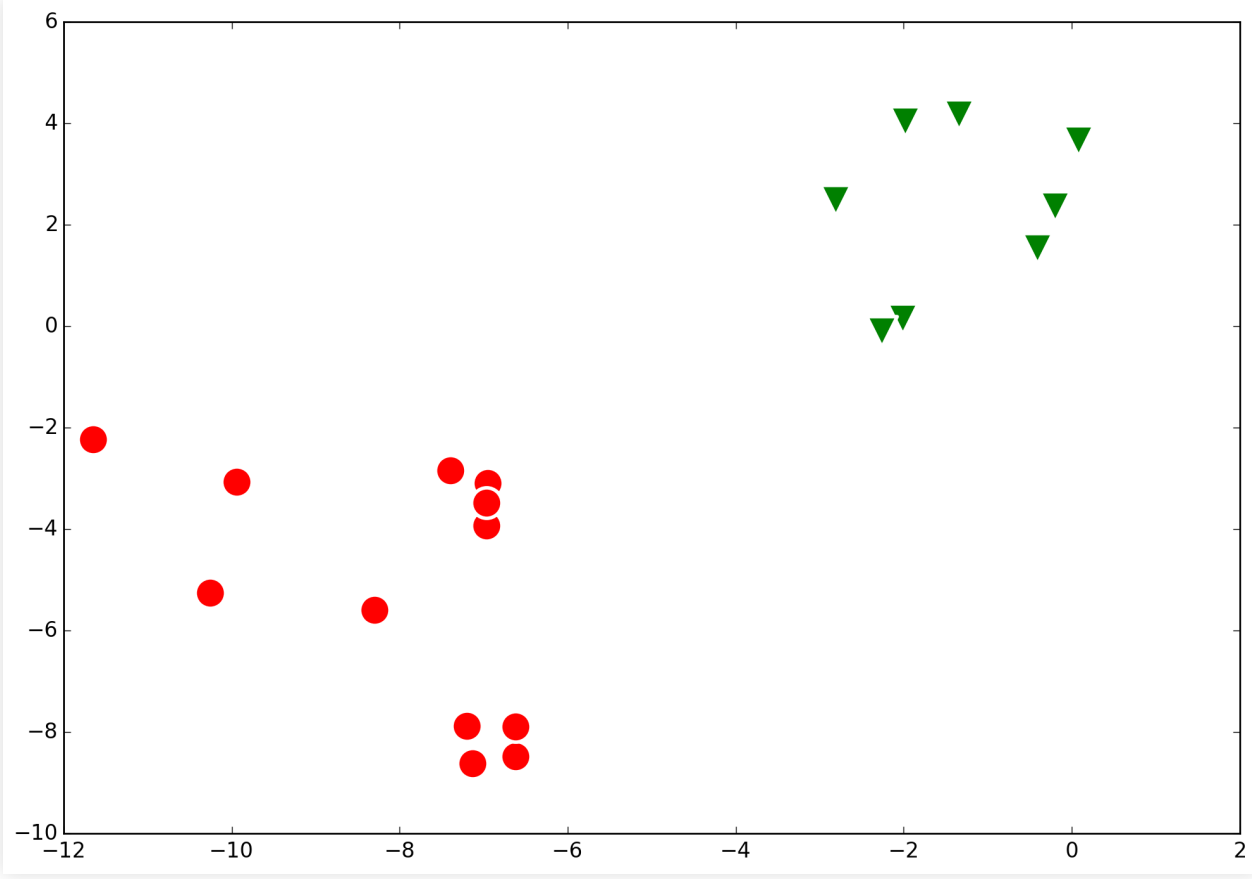
Why clustering?

To reduce (summarize) data:



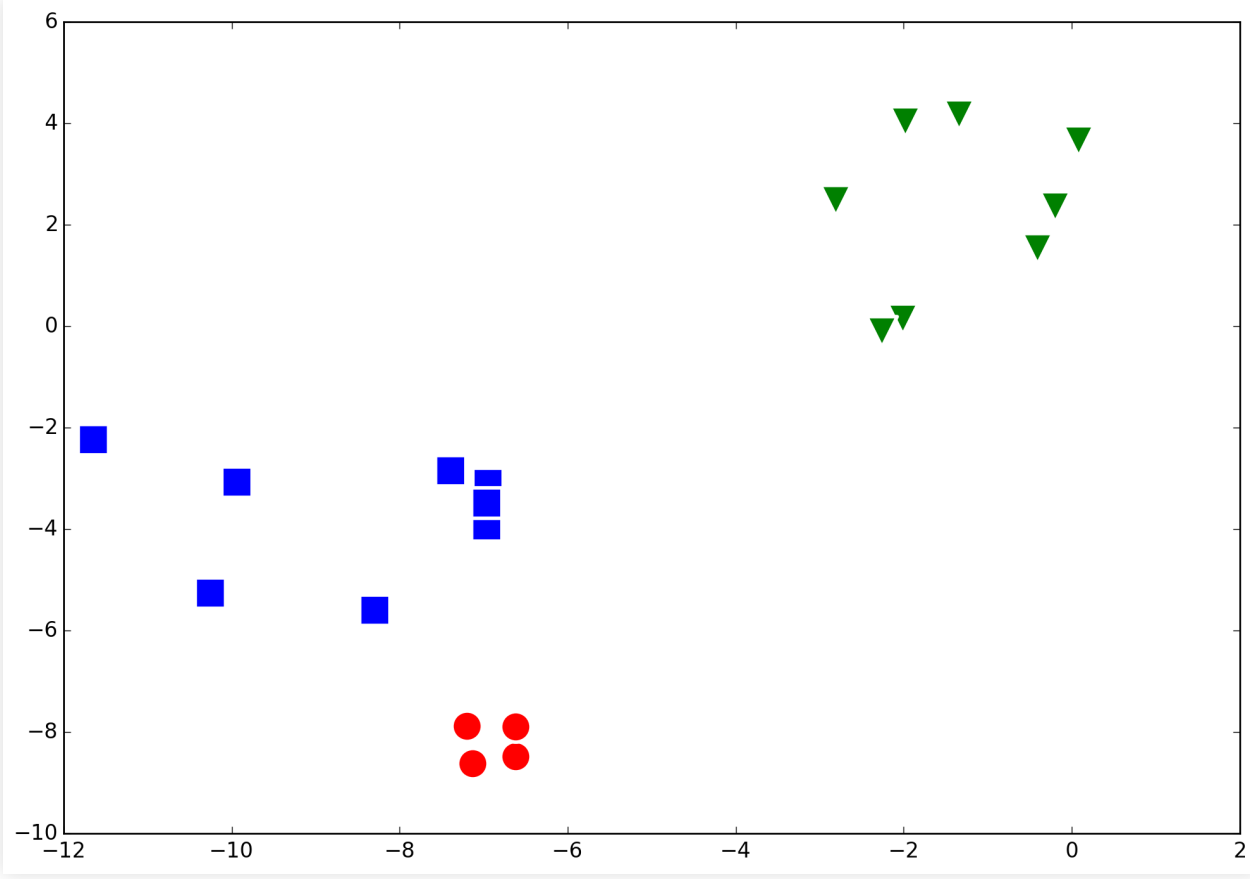
Defining clusters

- Are there two clusters here?



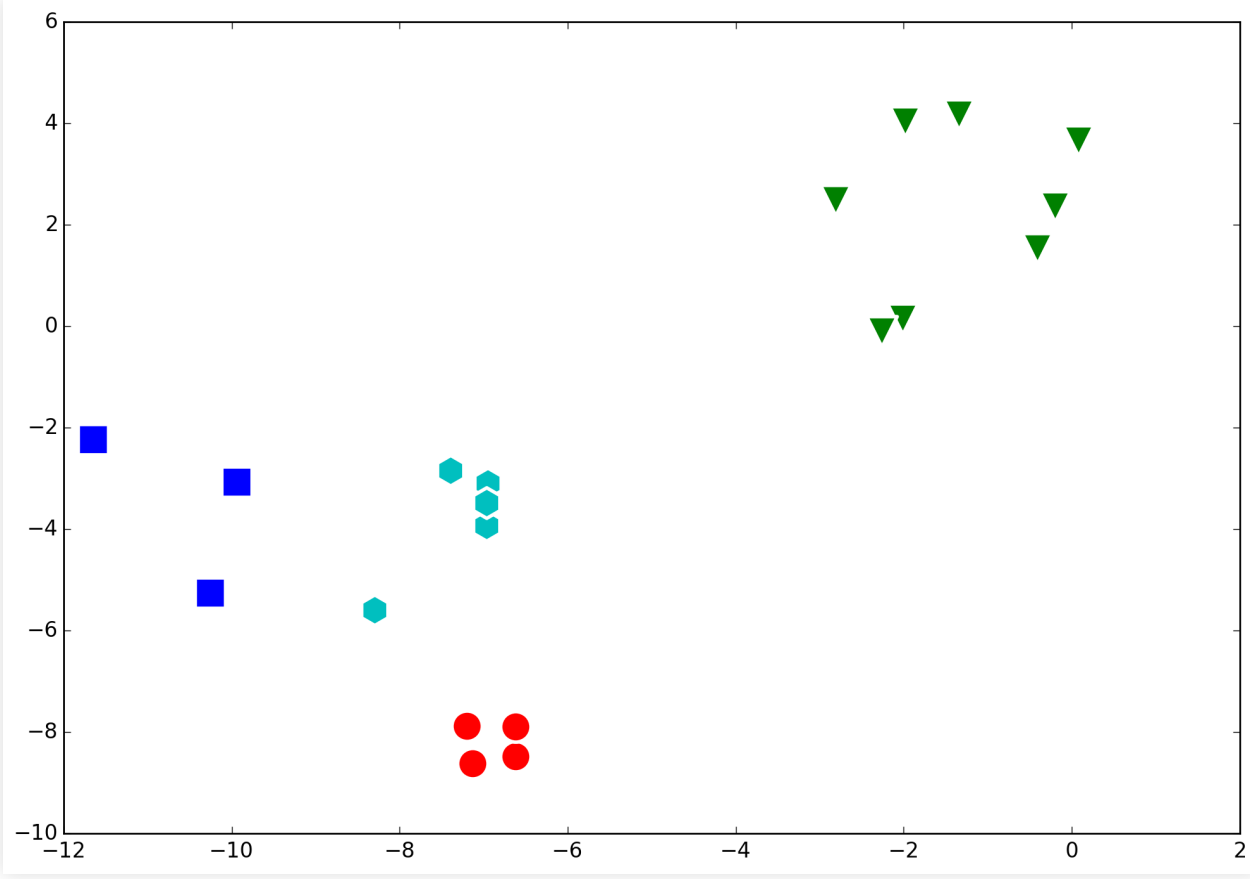
Defining clusters

- Are there three clusters here?



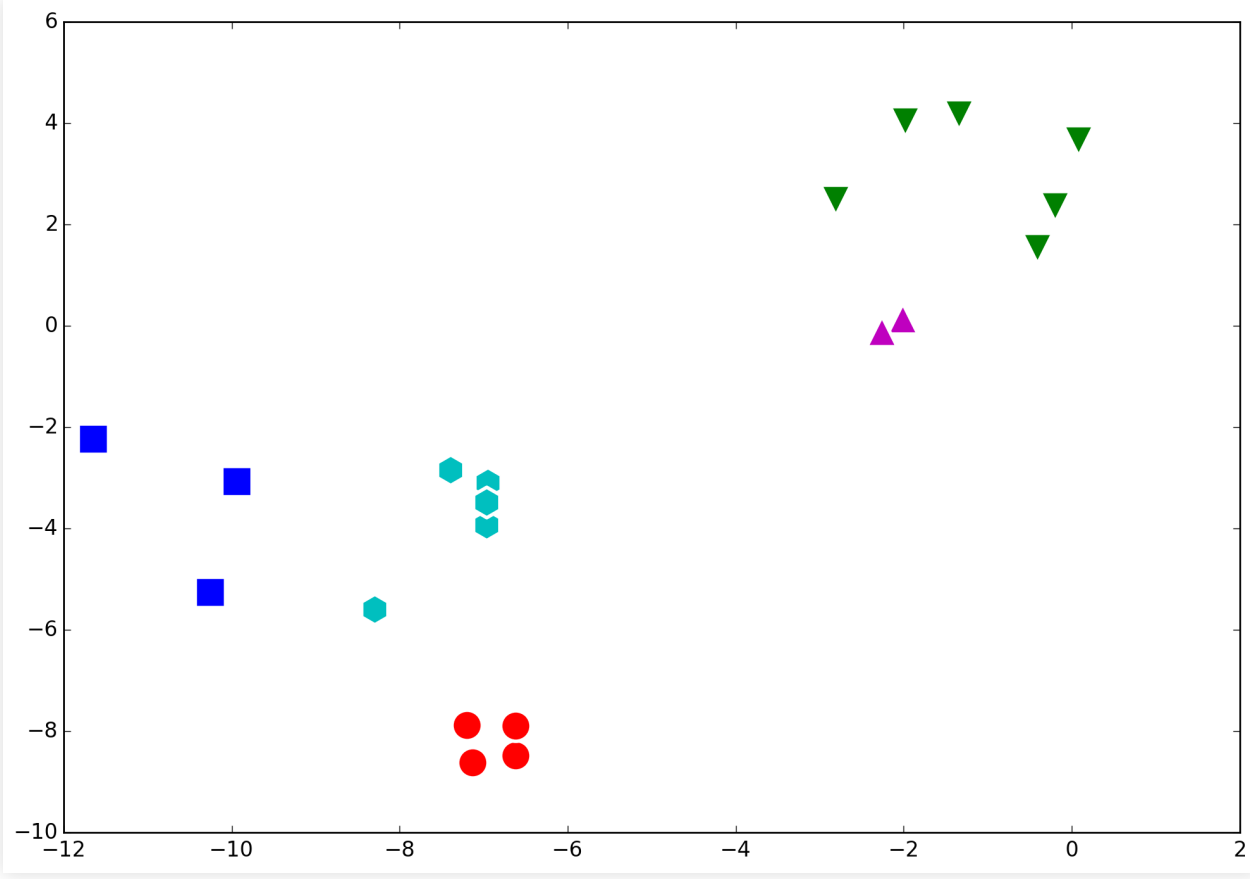
Defining clusters

- Are there four clusters here?



Defining clusters

- Are there five clusters here?



Defining clusters

How do we define our clusters?

- Several things we need to decide about the set of clusters
- Partitional clustering:

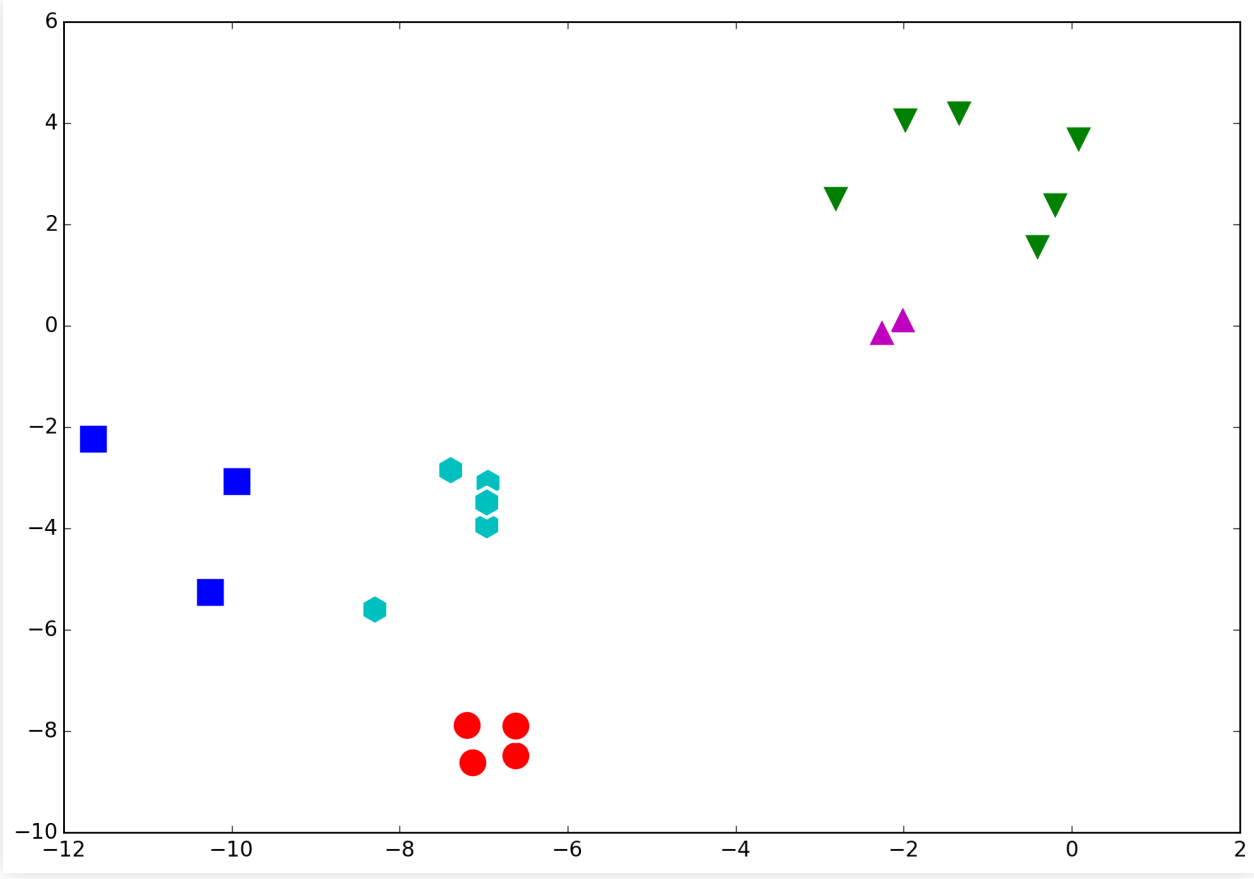
• Data is divided into groups at the same level

- Hierarchical clustering:

• Clusters are nested within larger clusters, in a tree

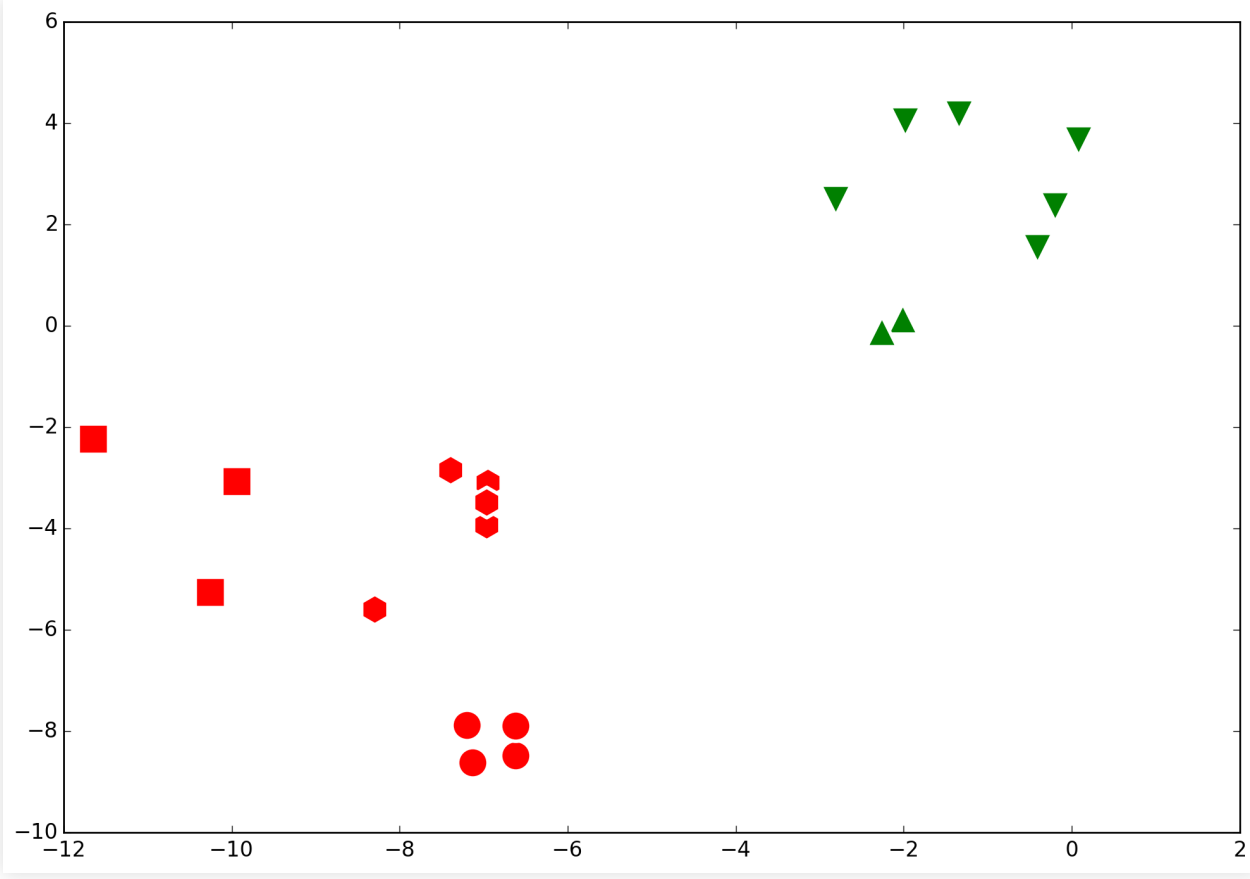
Defining clusters

- Partitional clustering



Defining clusters

- Hierarchical clustering



Clustering, example membership

- Exclusive clustering:
  - Each example belongs to only one cluster
- Overlapping clustering:
  - Examples may belong to more than one cluster
- Fuzzy clustering:
  - Each example belongs to clusters with $w_k \in [0; 1]$
- Probabilistic clustering:
  - As fuzzy, but $\sum_{k=1}^{K} w_k = 1$

Defining clusters

Clustering, coverage of examples

- Complete clustering:

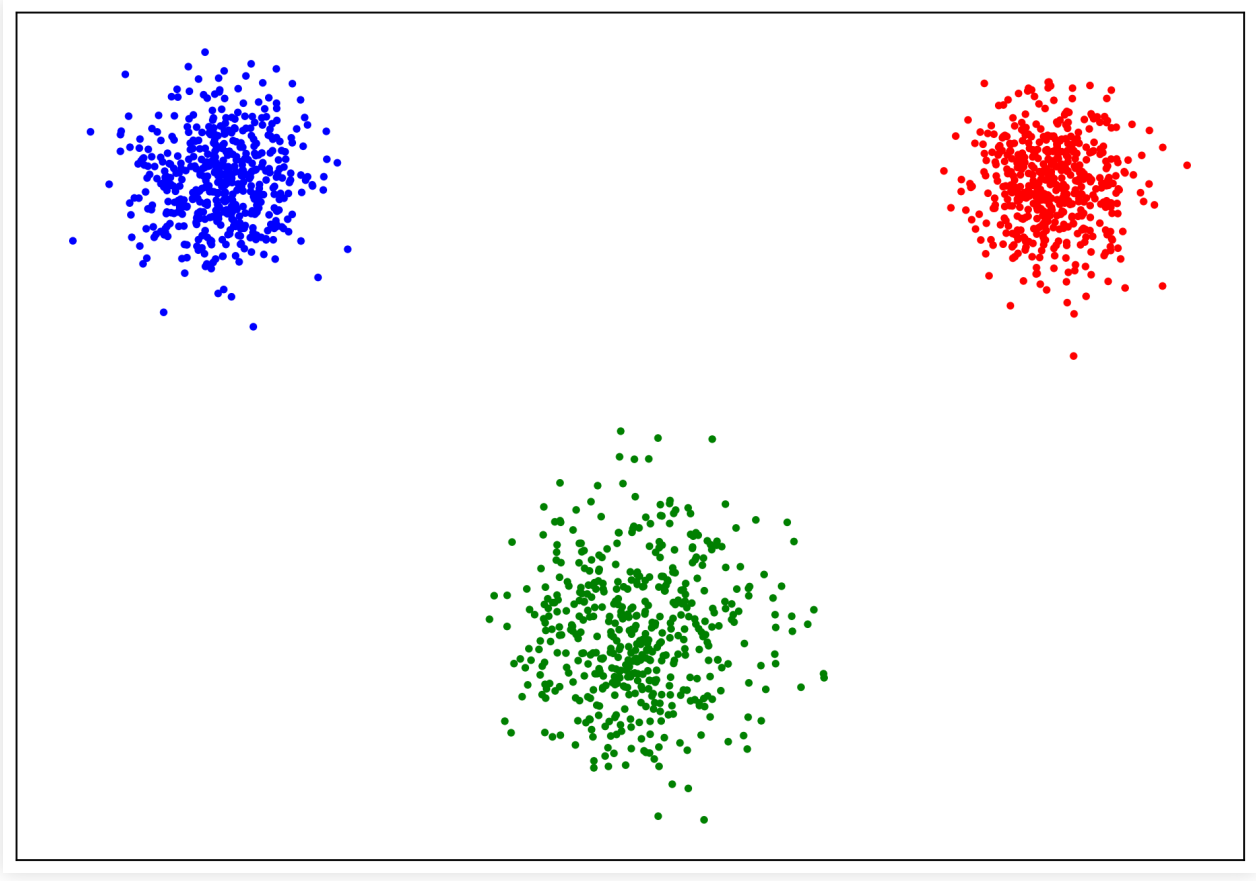
- All examples are assigned to cluster (or clusters)

- Partial clustering:

- Some examples unassigned (e.g. noise, irrelevant data, etc)

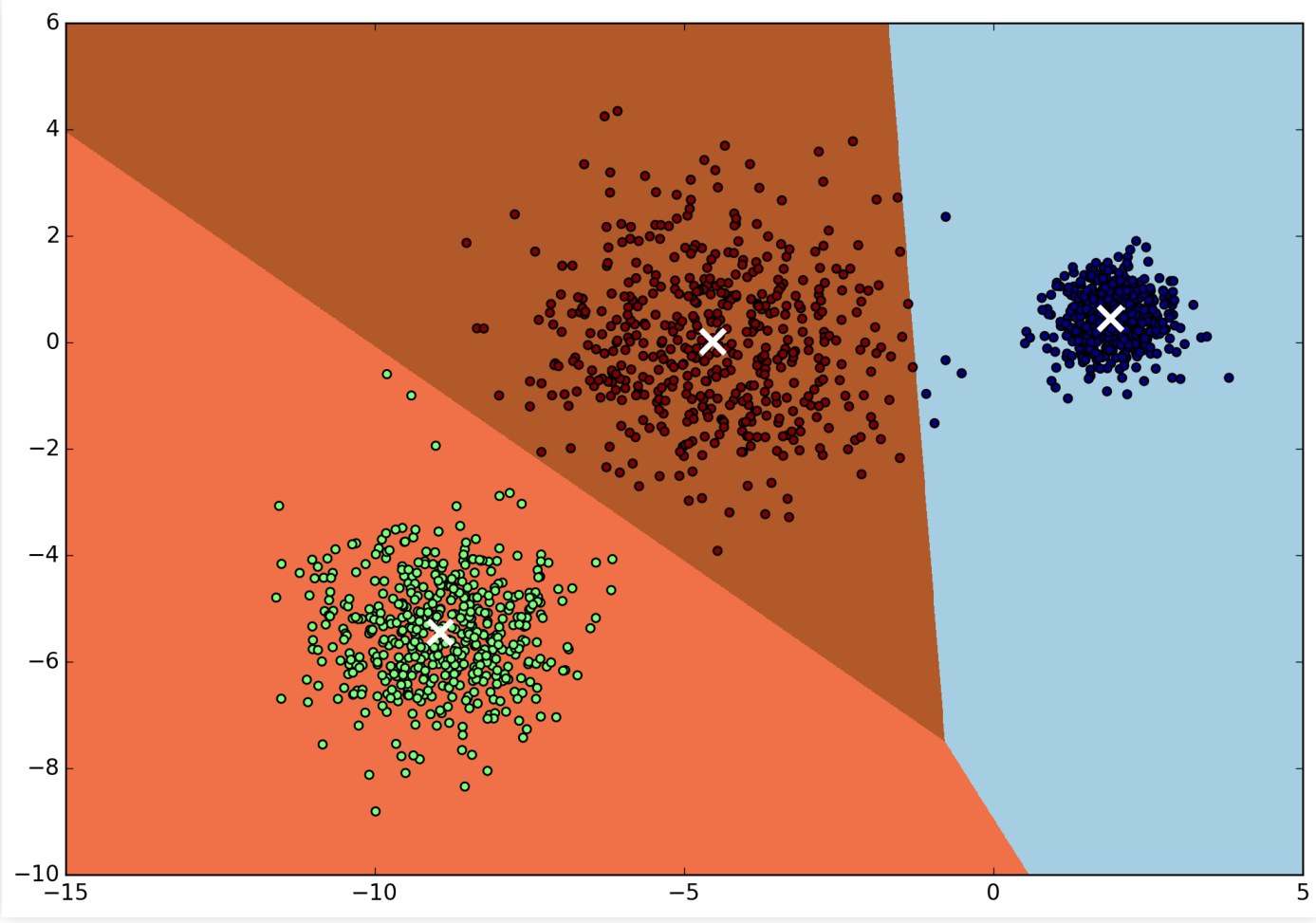
Types of clusters

- Well-Separated (data dependent): More similar to all cluster than other clusters



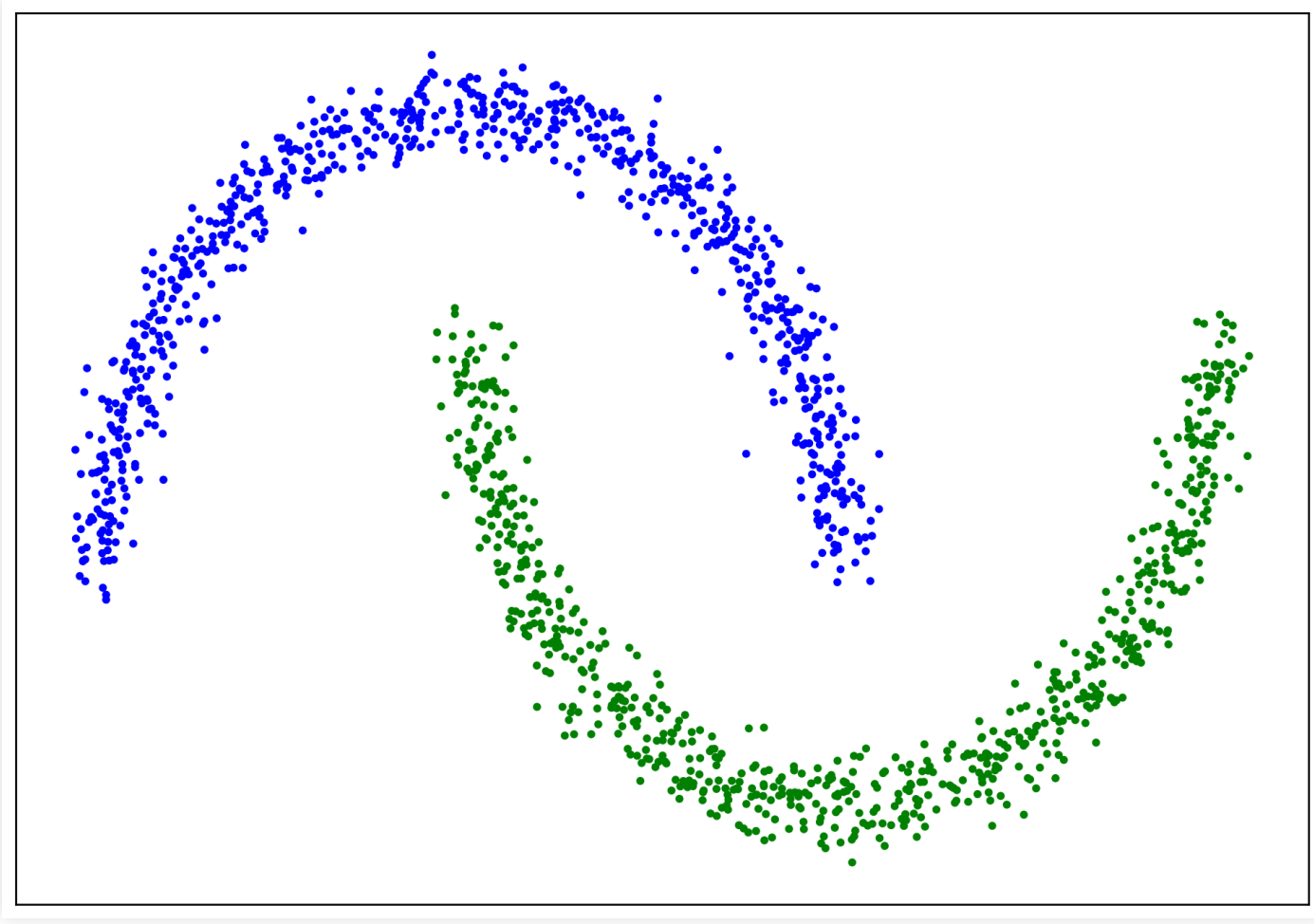
Types of clusters

- Prototype-based clustering: Examples closer to prototype



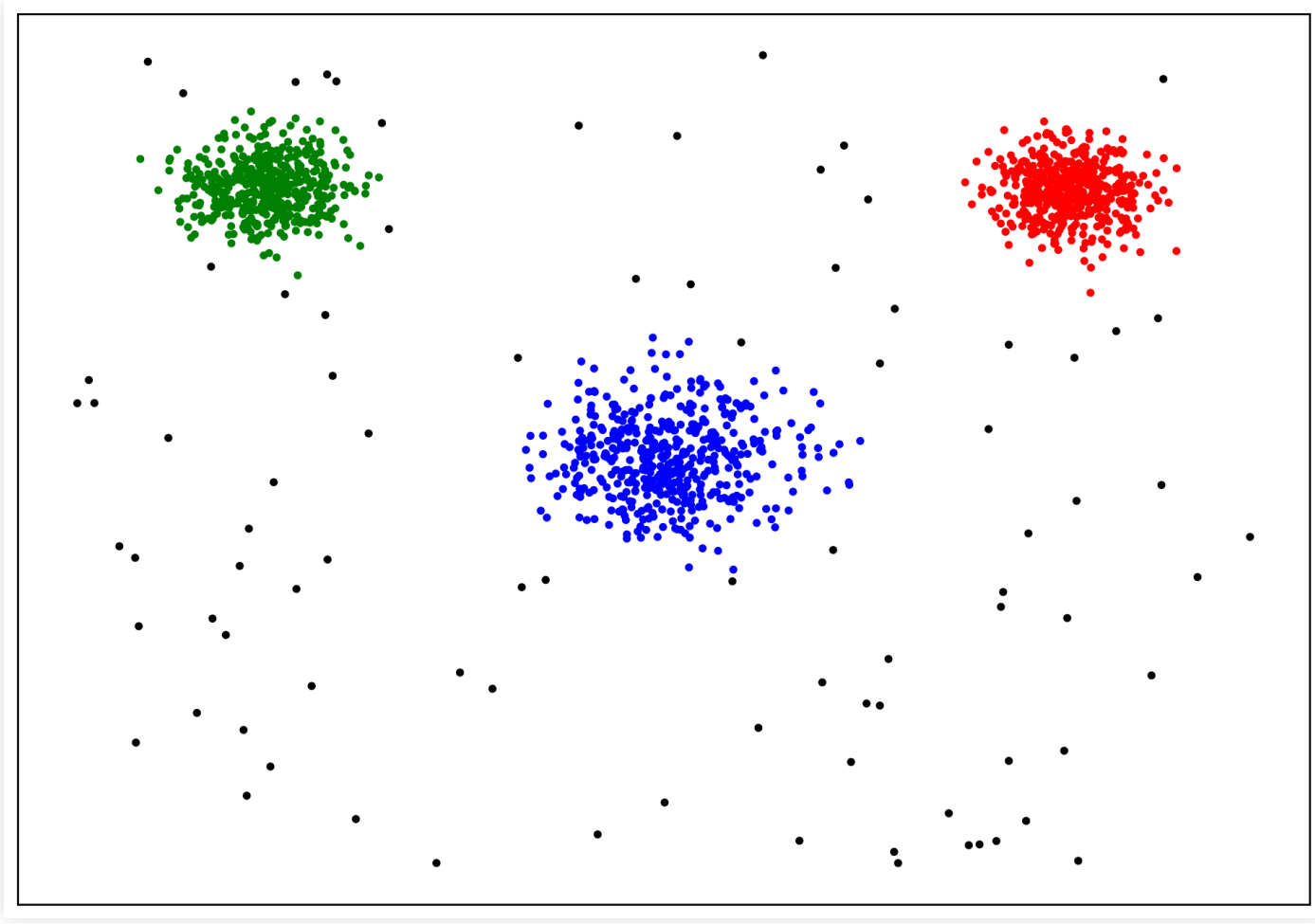
Types of clusters

- Contiguity-based clustering: Closer in cluster than outside



Types of clusters

- Density-based clustering:High density regions (discards noise)



Clustering

- How do we compare examples?
  - (What similarity measures?)
- How are examples related to clusters?
  - (All in clusters? Some? Partially or fully?)
- How do we group examples together?
  - (Prototypes? Density? Affinity?)
- How meaningful are the clusters?
  - (Statistically significant? Make sense?)

The goal: making data more intelligible

Clustering vs Classification

- In classification we have a clear goal:

- Predict the labels given by the data
- We can measure error (supervised learning)

- In clustering the goal is not defined a priori:

- Why do we want to cluster?
- Simplify, find meaningful groups, understand patterns?
- How do we want to cluster?

- In clustering the categories are given by the model and not the data.

- It's like classification but the we invent the classes...

K-MEANS

K-means

Find best clustering of k clusters

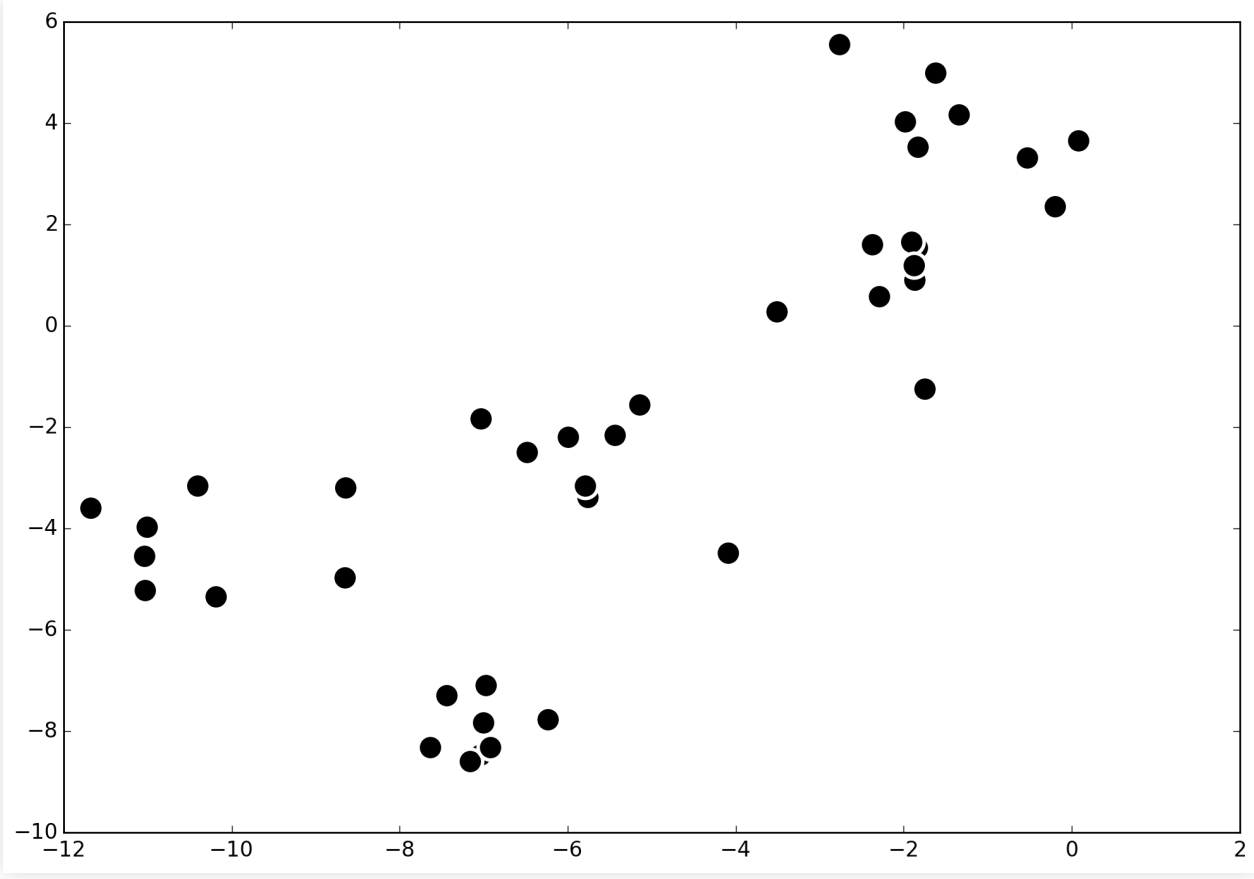
- Define clusters by proximity to the mean of the cluster
- The number of centroids and clusters is predefined (k)
- Partitional, exclusive, complete and prototype-based

K-means (Lloyd's) algorithm

- Start with random centroids
- Assign each example to closest centroid
- Update centroids to mean of respective cluster
- Recompute clusters, repeat until convergence

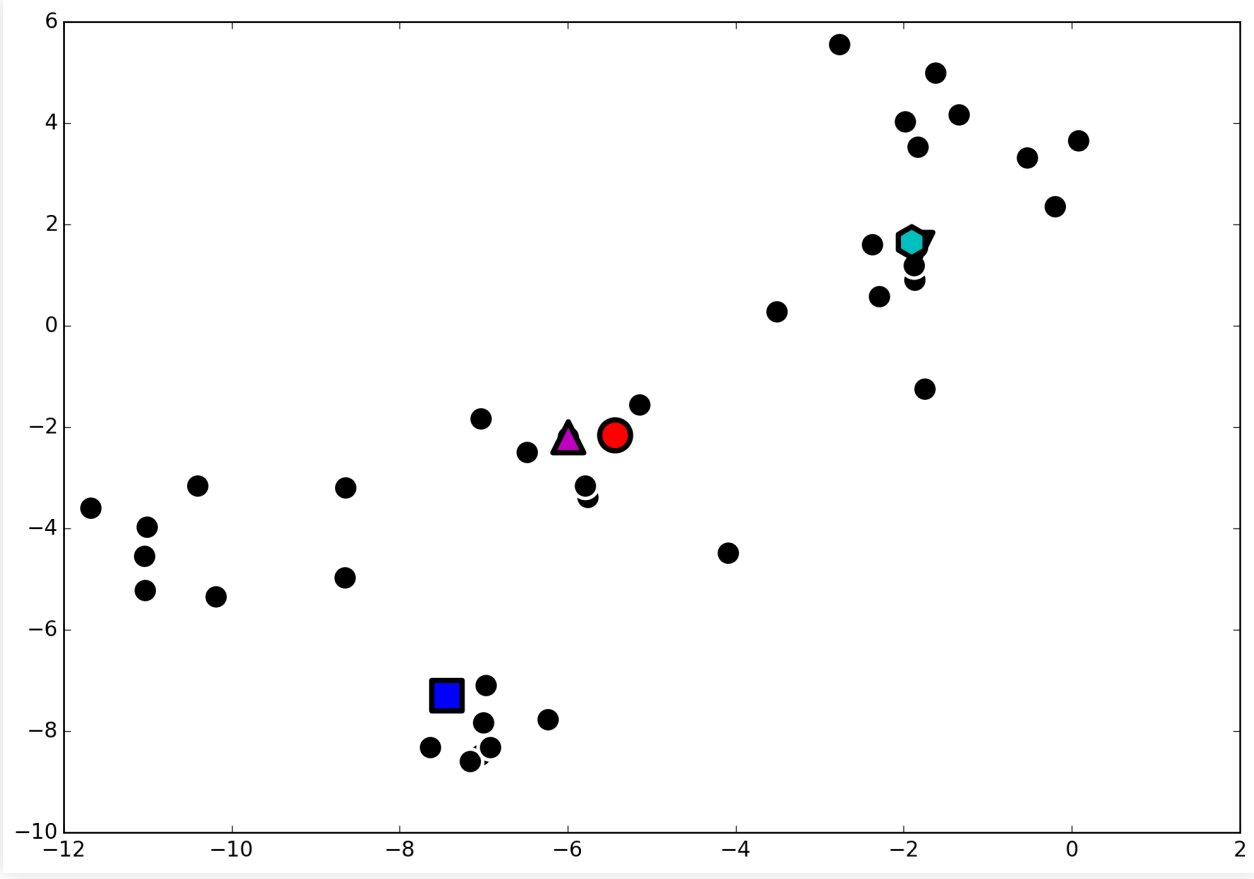
K-means initialization

- Forgy : start with cordinates of a random set of examples



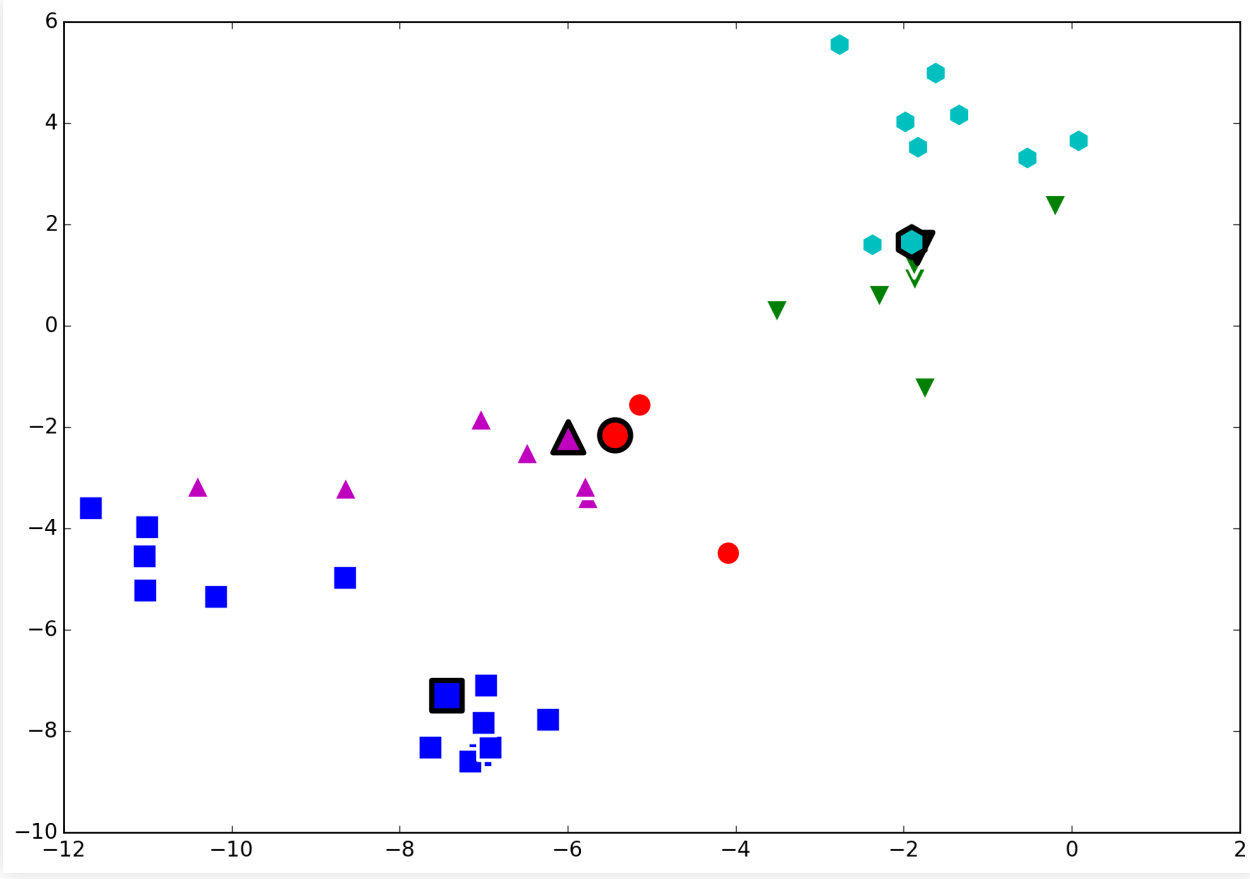
K-means initialization

- Forgy : start with cordinates of a random set of examples



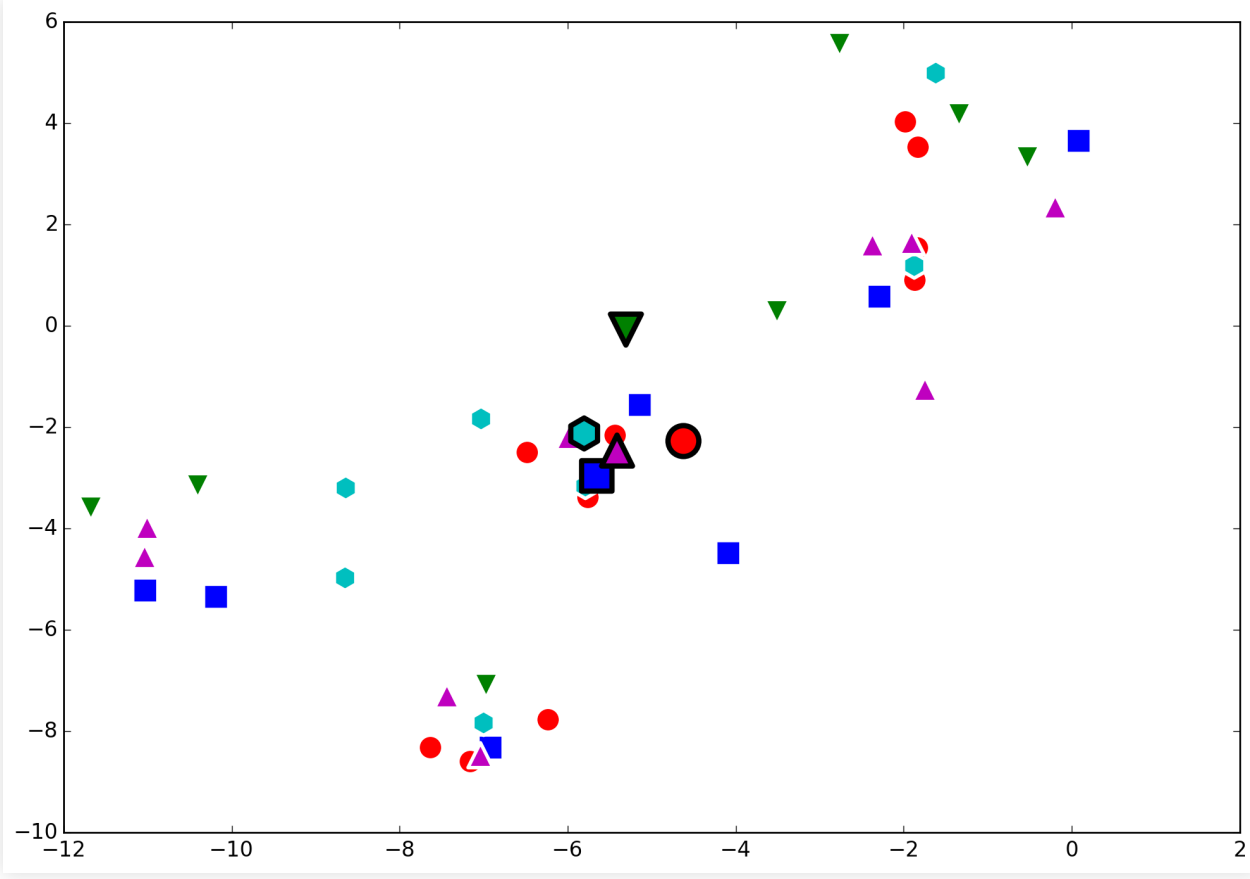
K-means initialization

- Forgy : assign each point to closest centroid



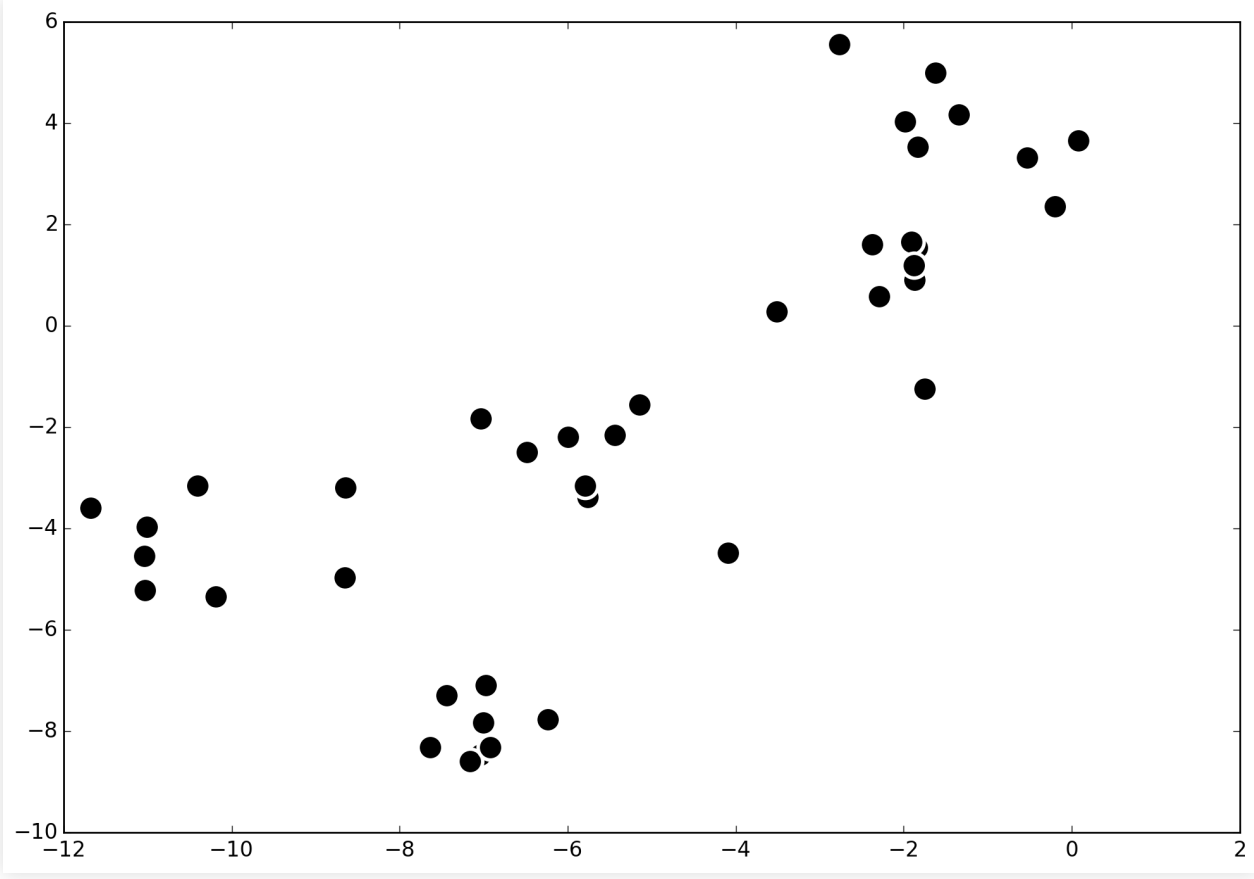
K-means initialization

- Random Partition : random assignment, compute means



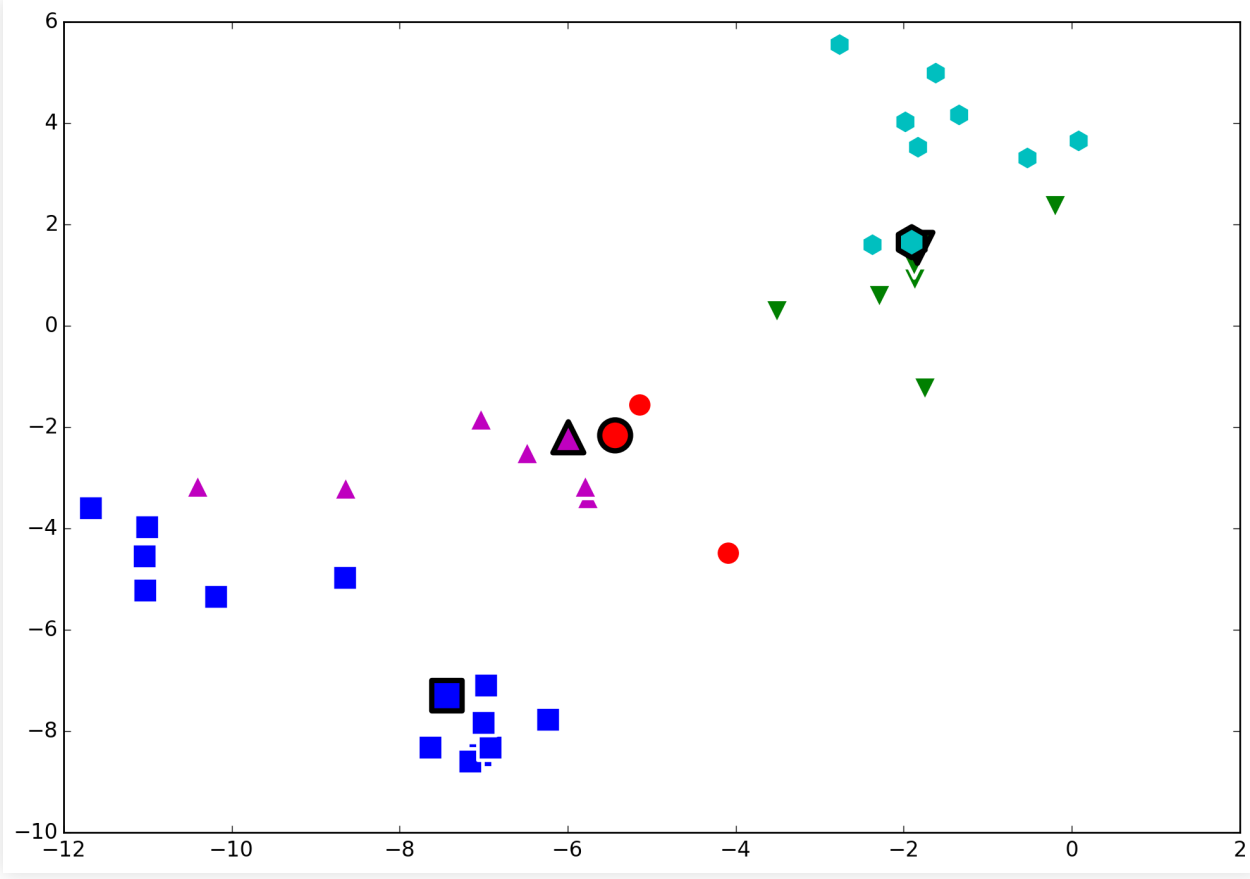
K-means algorithm

- Original data:



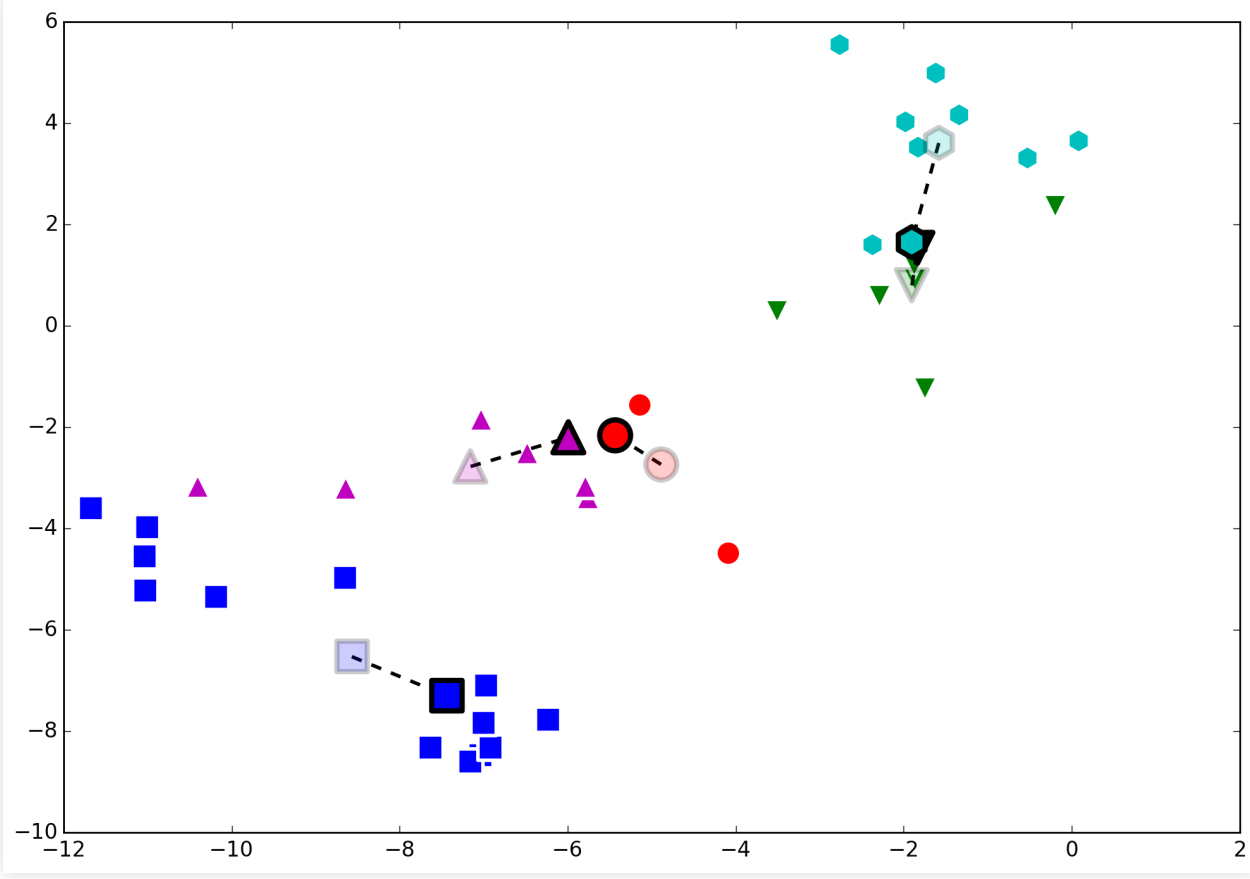
K-means algorithm

- Initialize (Forgy), compute clusters



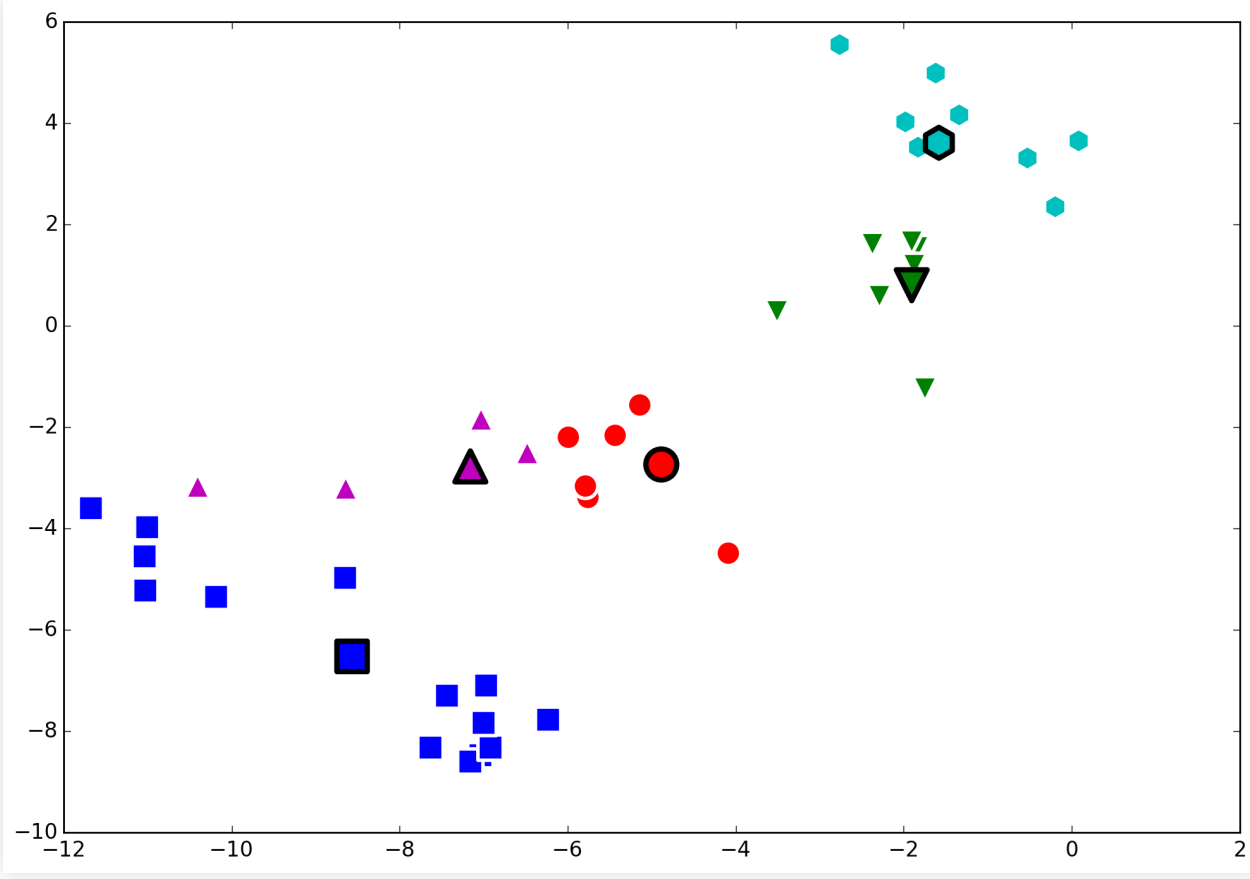
K-means algorithm

- Compute new means, update centroids



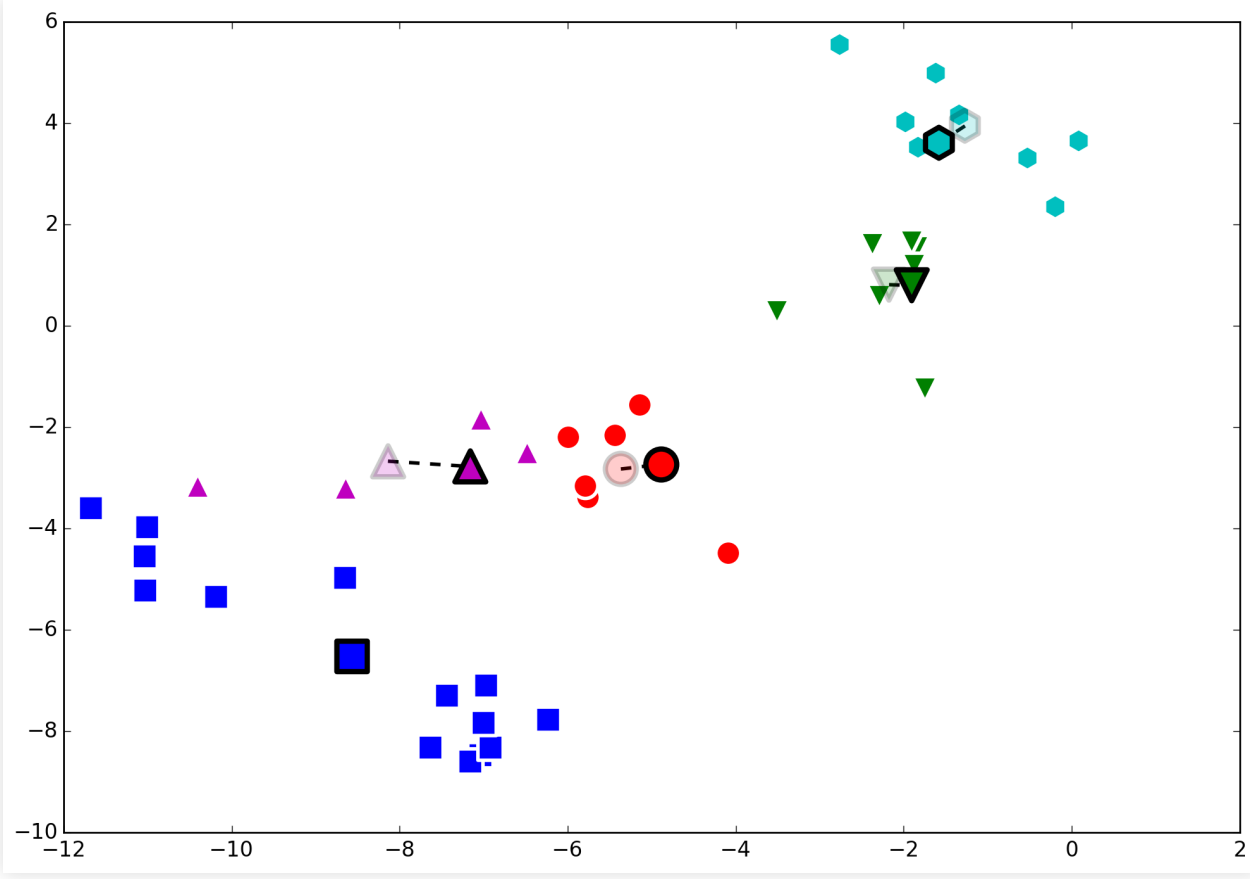
K-means algorithm

- Recompute clusters



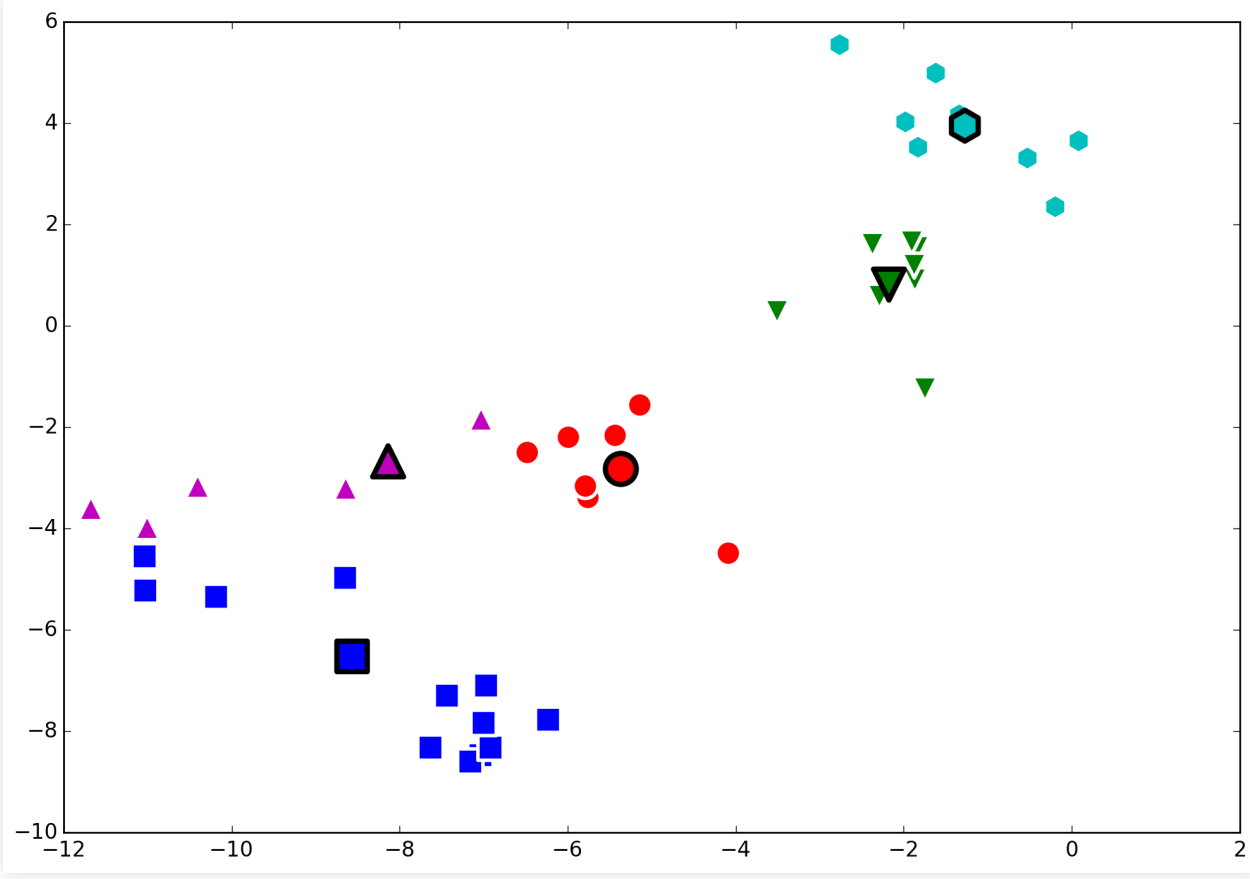
K-means algorithm

- Compute new means, update centroids



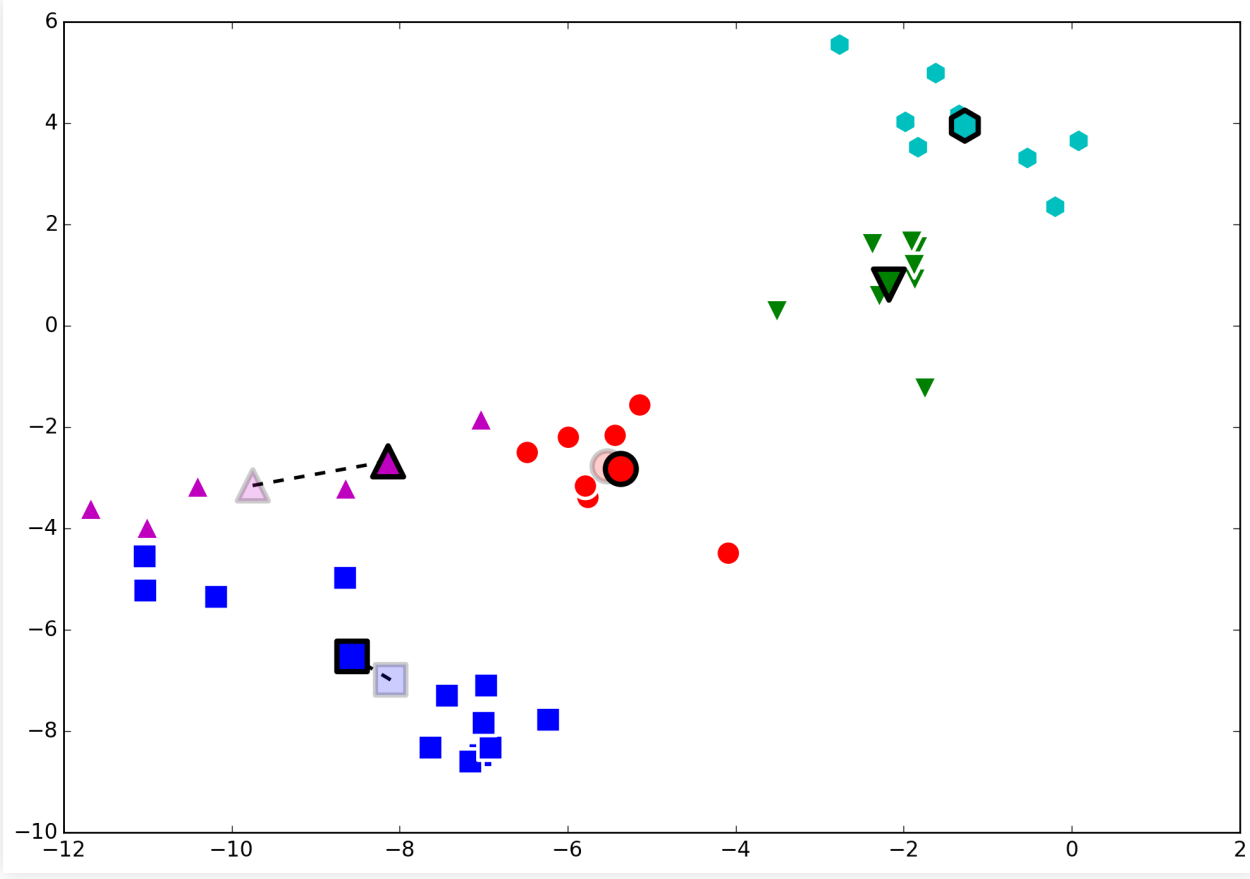
K-means algorithm

- Recompute clusters



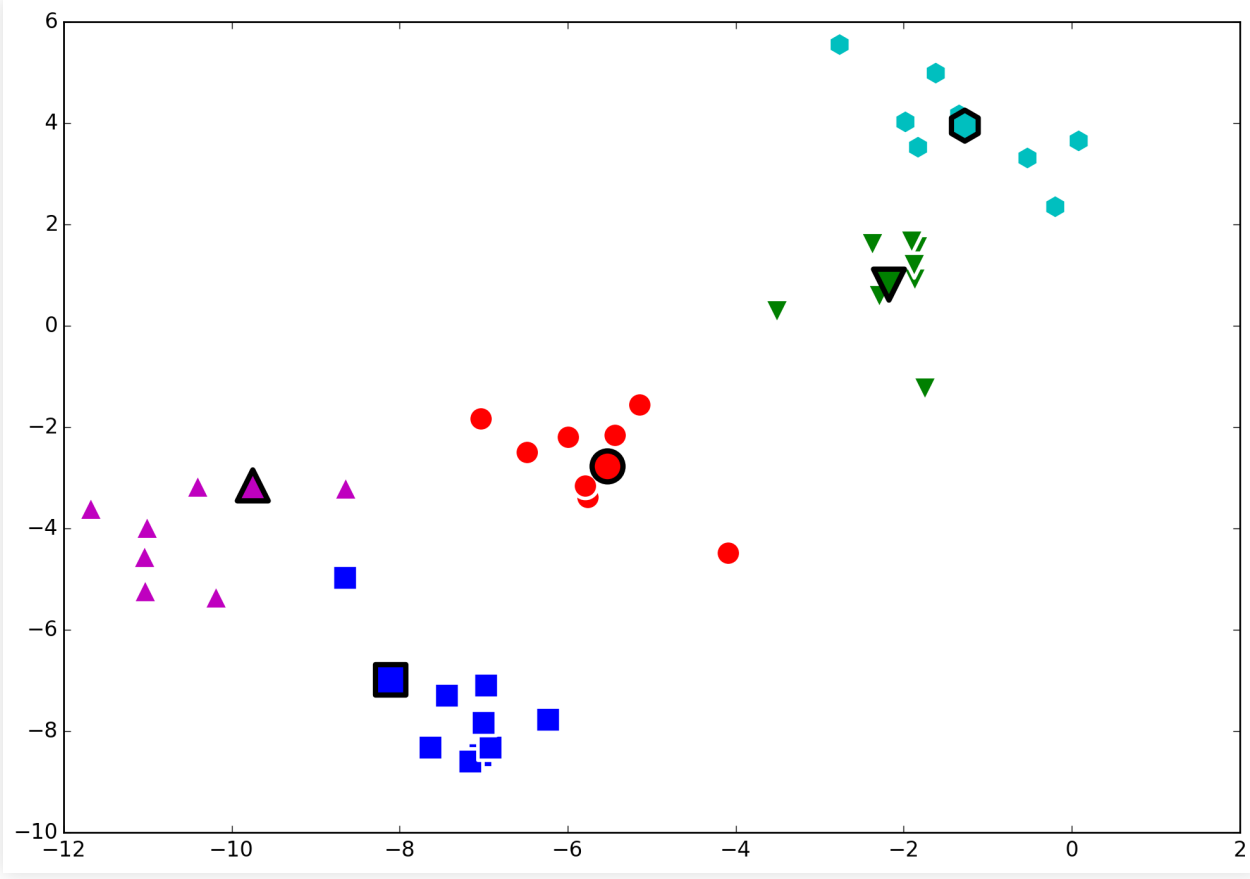
K-means algorithm

- Compute new means, update centroids



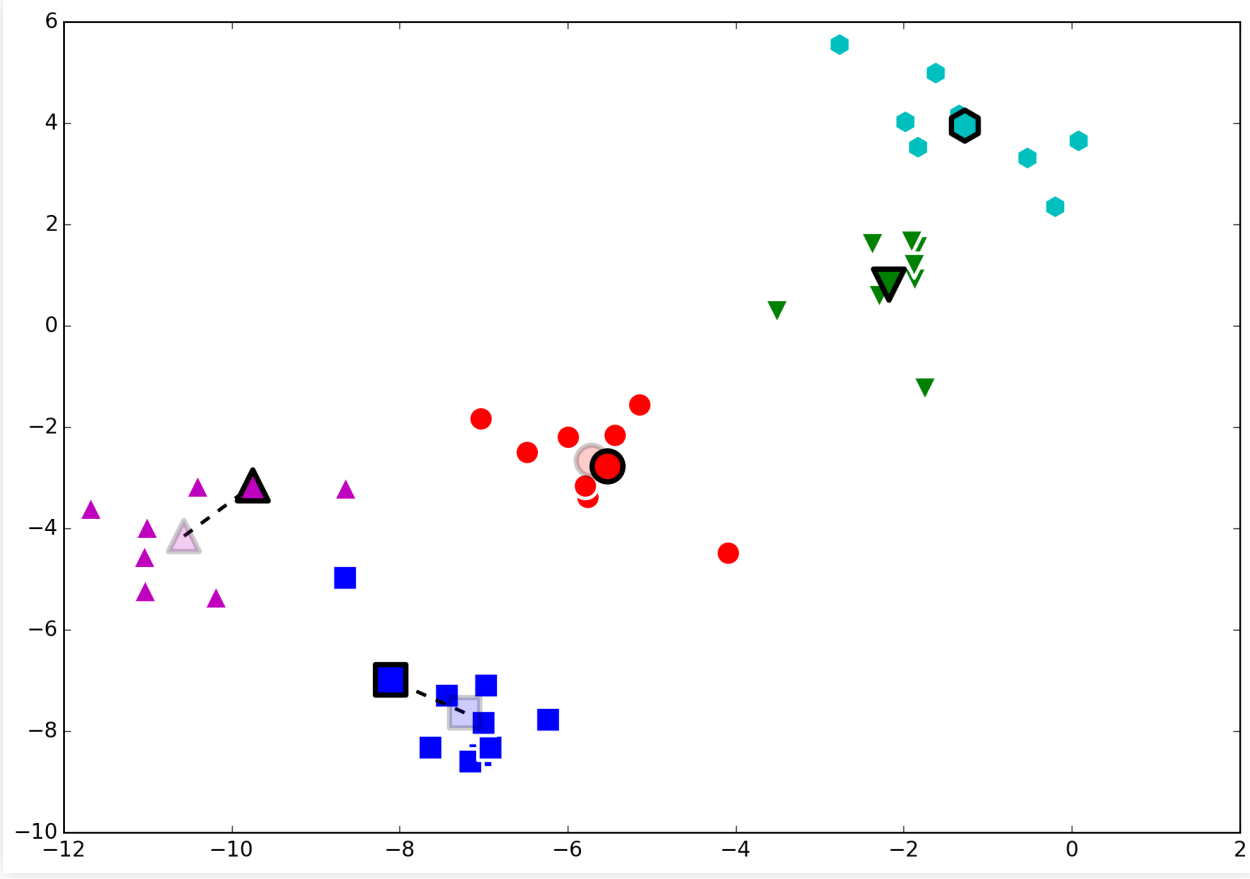
K-means algorithm

- Recompute clusters



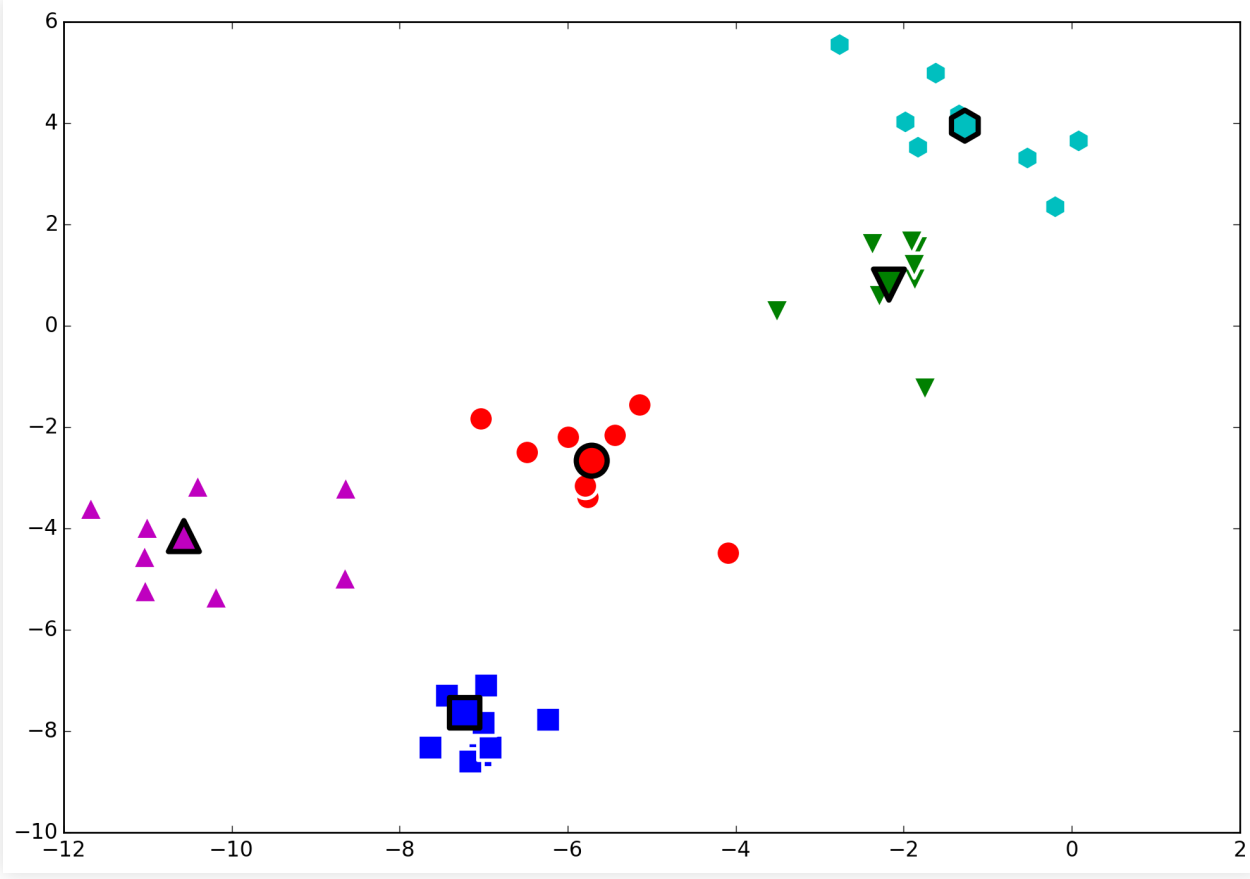
K-means algorithm

- Compute new means, update centroids



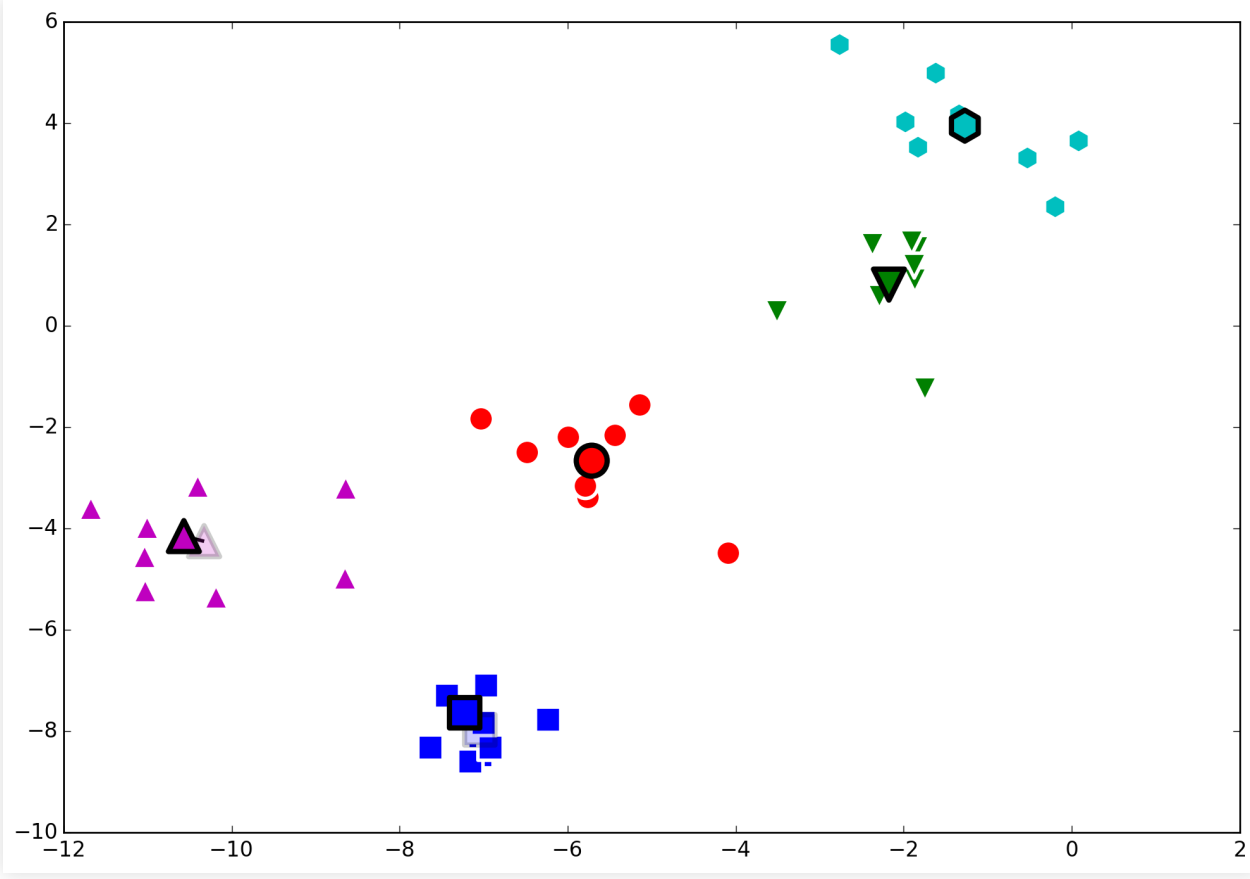
K-means algorithm

- Recompute clusters



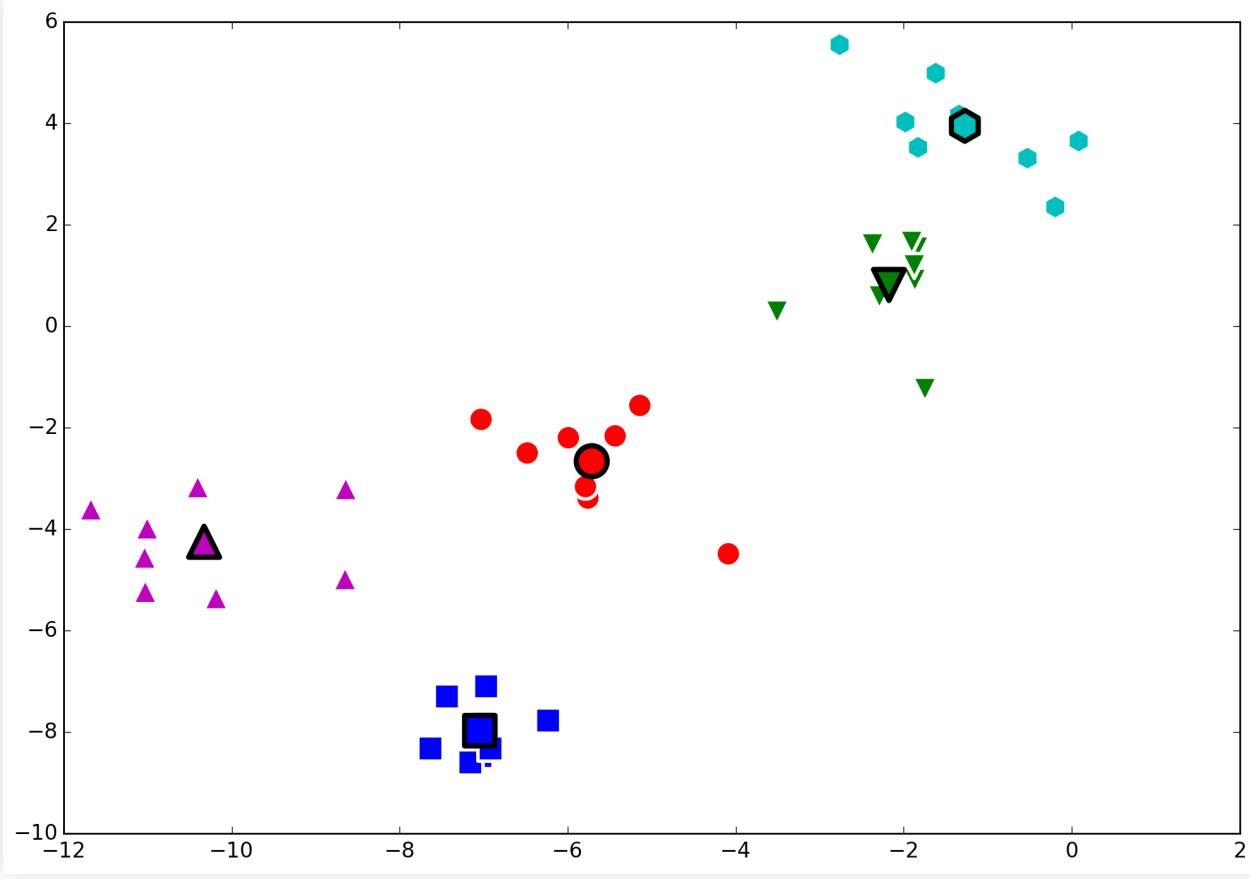
K-means algorithm

- Compute new means, update centroids



K-means algorithm

- Until convergence



K-means algorithm

- Attribute points to centroids

```
def closest_centroids(data,centroids):
    ys = np.zeros(data.shape[0])
    for ix in range(data.shape[0]):
        dists = np.sum((centroids-data[ix,:])**2,axis=1)
        ys[ix] = np.argmin(dists)
    return ys
```

- Move centroids to mean points

```
def adjust_centroids(data,centroids):
    ys = closest_centroids(data,centroids)
    for ix in range(centroids.shape[0]):
        centroids[ix,:] = np.mean(data[ys==ix,:],axis=0)
```

K-means algorithm

- Initialization (Forgy method)

```
def forgy(data,k):
    ixs = np.arange(data.shape[0])
    np.random.shuffle(ixs)
    return data[ixs[:k],:].copy()
```

- Initialization (Random Partition)

```
def rand_part(data,k):
    ys = np.random.randint(0,k,data.shape[0])
    centroids = np.zeros((k,data.shape[1]))
    for ix in range(k):
        centroids[ix,:] = np.mean(data[ys==ix,:],axis=0)
    return centroids,ys
```

K-means algorithm

- Finding the best prototypes for the clusters
- Distance measure, typically euclidean, but Minkowsky more general:

$$D_{x,x'} = \sqrt[p]{\sum_d |x_d - x'_d|^p}$$

- Conceptually similar to a SOM (without the topology)

K-medoids

- Variant of k-means where prototypes are data points (medoids)
- Needs only a dissimilarity matrix
- Assign each example to cluster of most similar medoid
- For each medoid and example, test if example is better medoid
- Stop when no improvement possible

EM algorithm

Expectation-Maximization algorithm

Background

- Set X of observed data
- Set Z of missing data or latent variables
- Unknown parameters θ
- Likelihood function $L(\theta; X, Z) = p(X, Z|\theta)$

Contrast with supervised learning:

- In classification classes are determined by observation

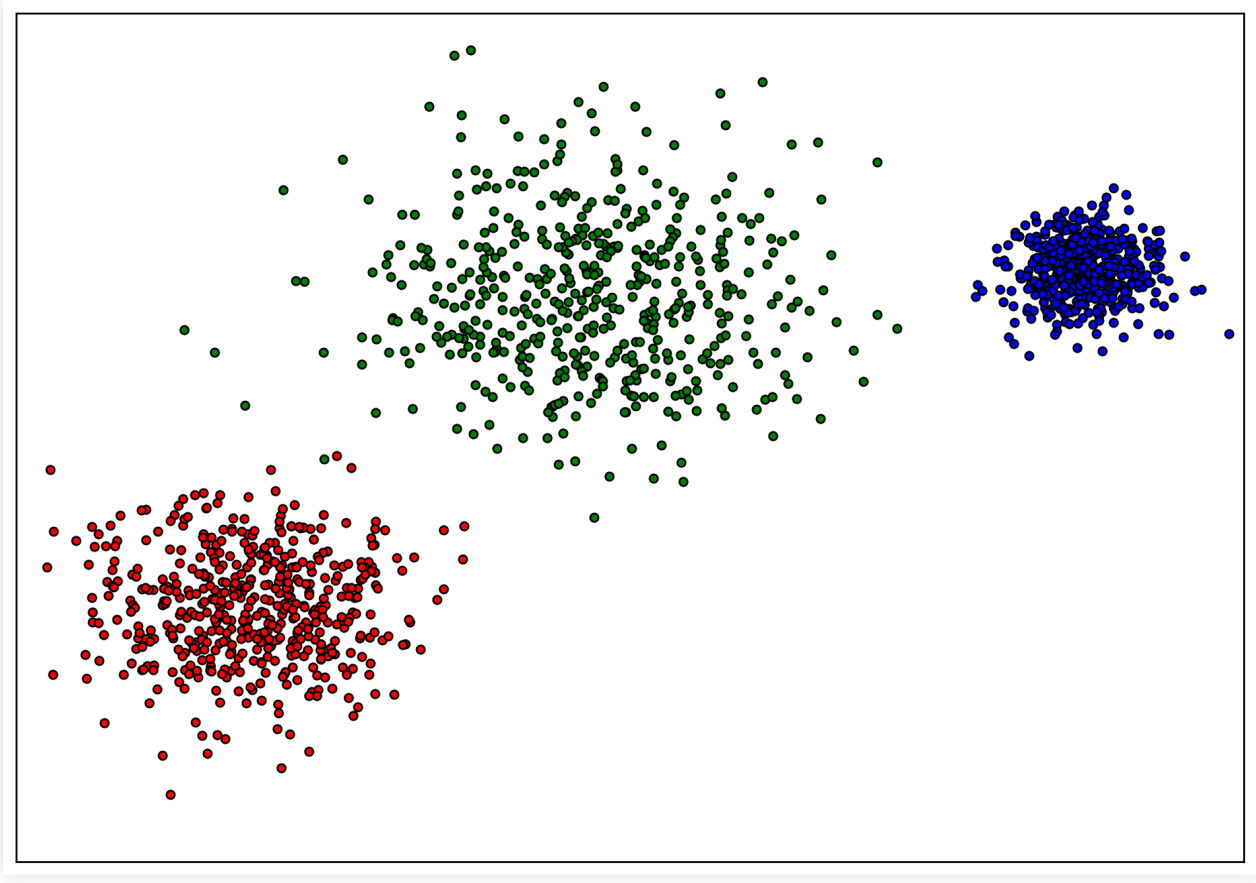
- Classes are labels in the training set

- In clustering, "classes" are latent variables in the model

- Clustering is like classification but with unspecified classes
- The algorithm will discover (invent?) the "classes"

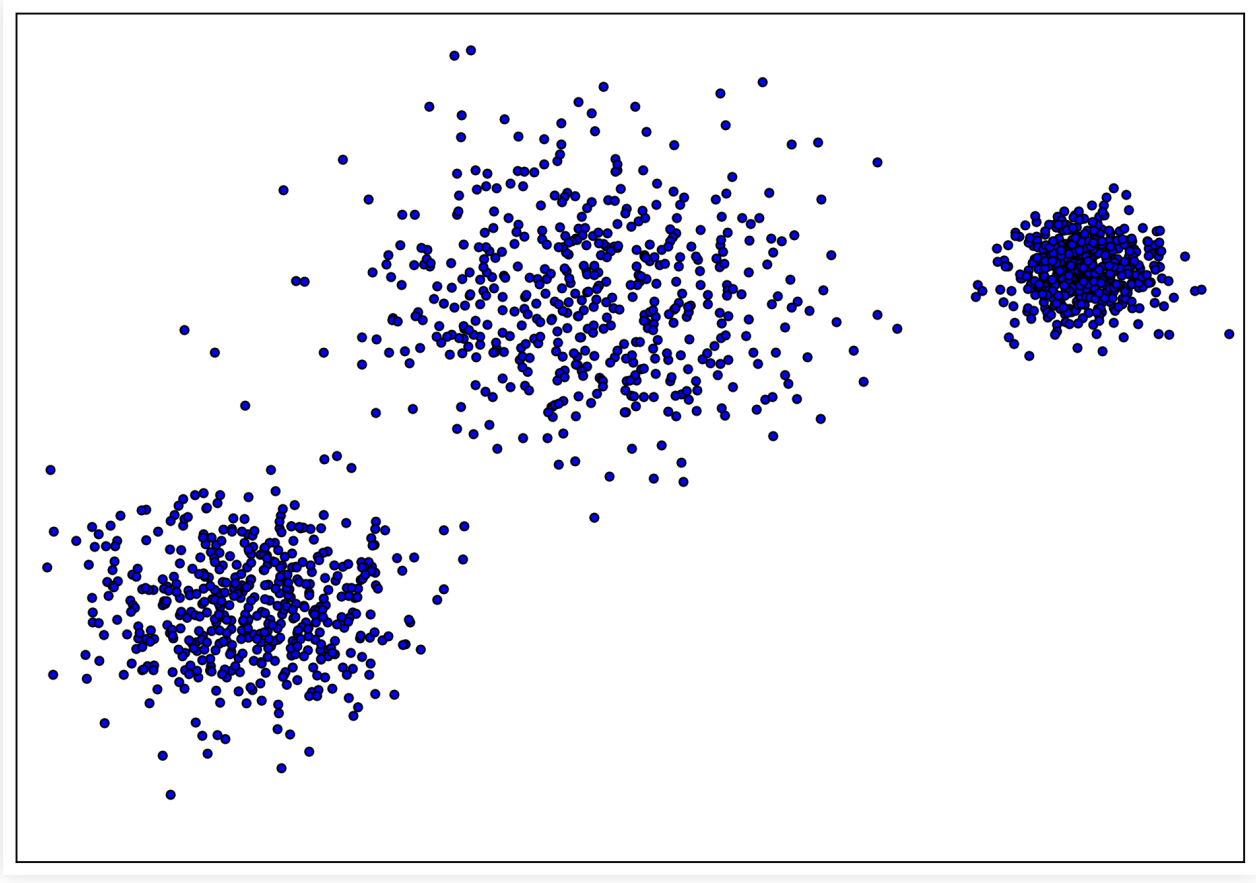
Expectation-Maximization algorithm

- Example: $\{X, Z\}$ is the complete data



Expectation-Maximization algorithm

- But we only have X, the incomplete data



Expectation-Maximization algorithm

- We cannot compute $L(\theta; X, Z) = p(X, Z|\theta)$ directly
- Because Z is unknown
- But we can estimate the posterior probability $p(Z|X, \theta^{old})$

EM algorithm

- Expectation step: compute the expected values of Z to get a likelihood function:

$$\mathcal{Q}(\theta, \theta^{old}) = \mathrm{E}_{Z|X,\theta^{old}} \ln p(X, Z|\theta)$$

- Maximization step: obtain θ^{new} maximizing $\mathcal{Q}(\theta, \theta^{old})$

$$\theta^{new} = \arg\max_{\theta} \mathrm{E}_{Z|X,\theta^{old}} \ln p(X, Z|\theta)$$

- Repeat $\theta^{old} \leftarrow \theta^{new}$ until convergence

EM and k-means

- We can see the EM idea in K-Means as an alternating optimization over different variables
- Define this distortion measure (objective function):

$$J = \sum_{n=1}^{N} \sum_{k=1}^{K} r_{nk} \|x_n - \mu_k\|^2$$

- Where r_{nk} is 1 if x_n belongs to μ_k, else 0
- We minimize first w.r.t. r_{nk} by assigning each example to the closest centroid
- Then we minimize w.r.t. each μ_k by setting the derivative to zero:

$$2 \sum_{k=1}^{K} r_{nk}(x_n - \mu_k) = 0 \iff \mu_k = \frac{\sum r_{nk} x_n}{\sum r_{nk}}$$

EM and k-means

- EM is (almost) the algorithm used for k-means

- Estimate r_{nk}, cluster assignment, from current centroids
- Estimation of the likelihood function (in this case, quadratic distance w.r.t. μ_k)
- Means maximize the estimated likelihood function (minimize distance)
- (We'll see this better after gaussian mixtures)

K-means Example

K-means color quantization

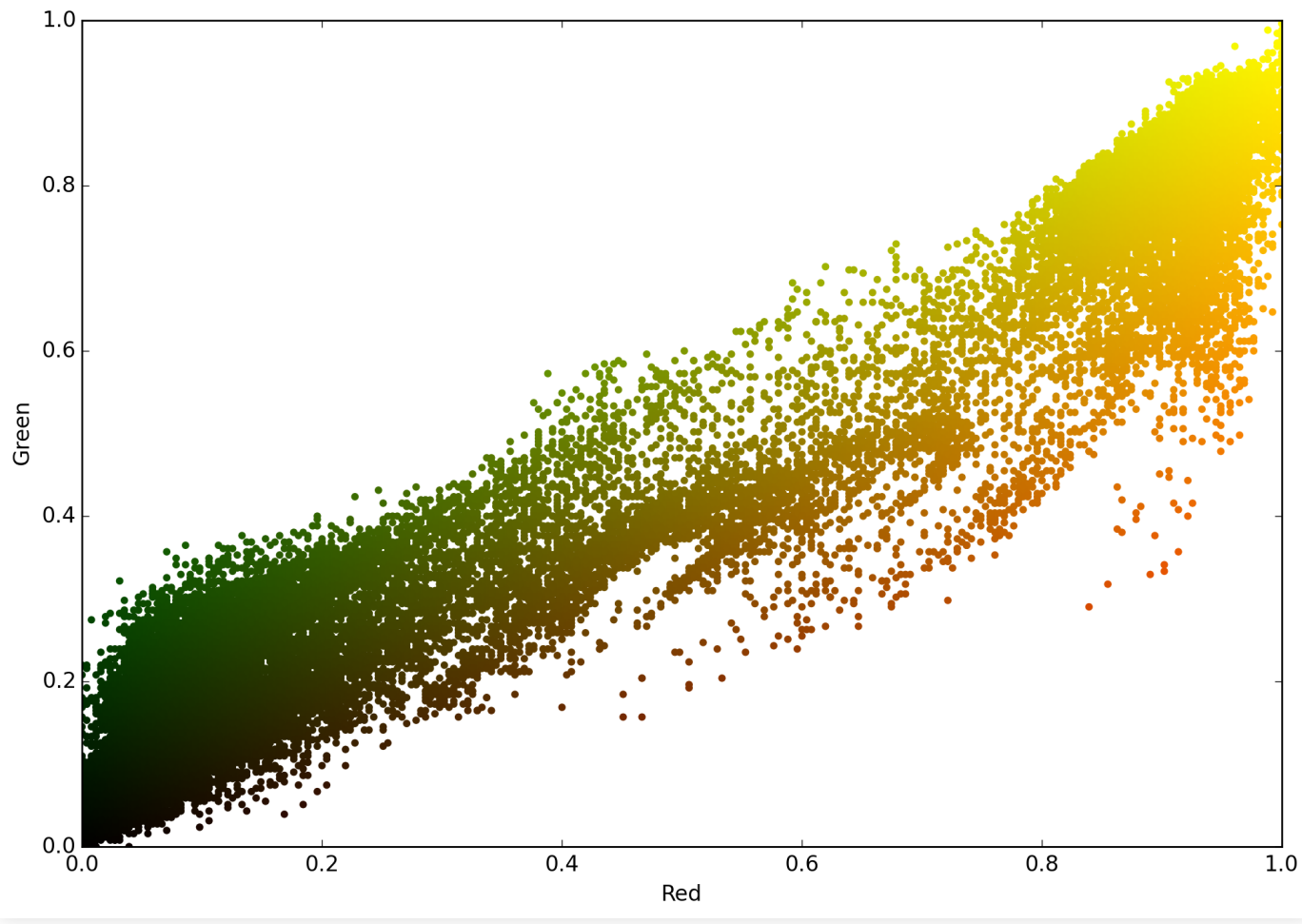
- Vector quantization is a common technique in signal compression
- We'll use k-means to quantize the color values of an image to reduce the color space
- Example image from Wikipedia
- Demo based on Scikit-learn demo on color quantization

K-means color quantization

- This image has only red and green (no blue)

K-means color quantization

- So we can plot all pixels green vs red



K-means color quantization

- Load and create pixel array (red, green)

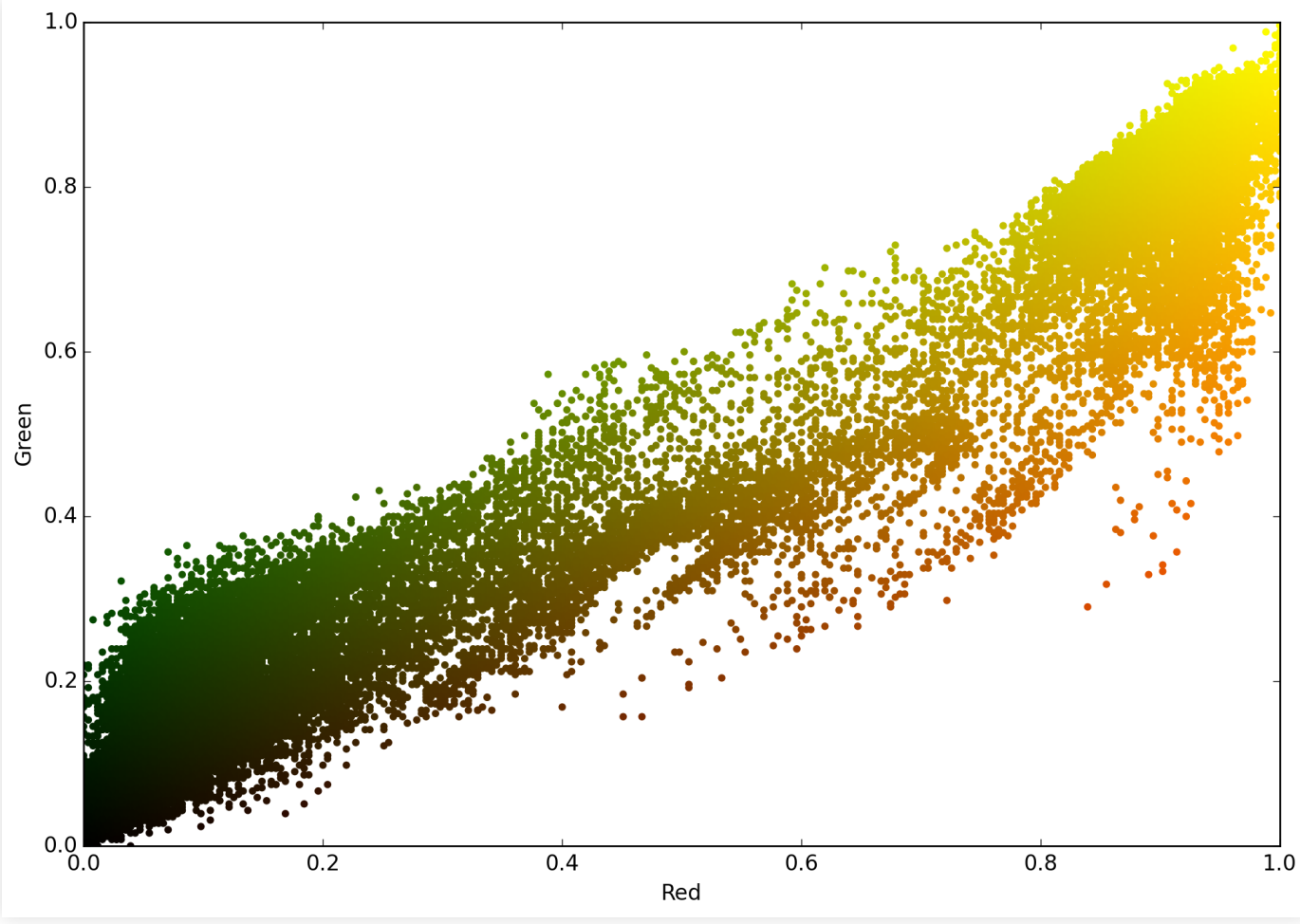
```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.cluster import KMeans
from skimage.io import imsave,imread

rosa = imread("Rosa.png")
w,h,d = rosa.shape
cols = np.reshape(rosa/255.0, (w * h, d))
image_array = cols[:,:2]
```

- Plot the pixel colors(red, green)

```
plt.figure(figsize=(12,8))
plt.xlabel('Red')
plt.ylabel('Green')
plt.scatter(image_array[:, 0], image_array[:, 1], color=cols.tolist(), s=10)
plt.axis([0,1,0,1])
plt.savefig('L17-rosa-plot.png', dpi=200,bbox_inches='tight')
```

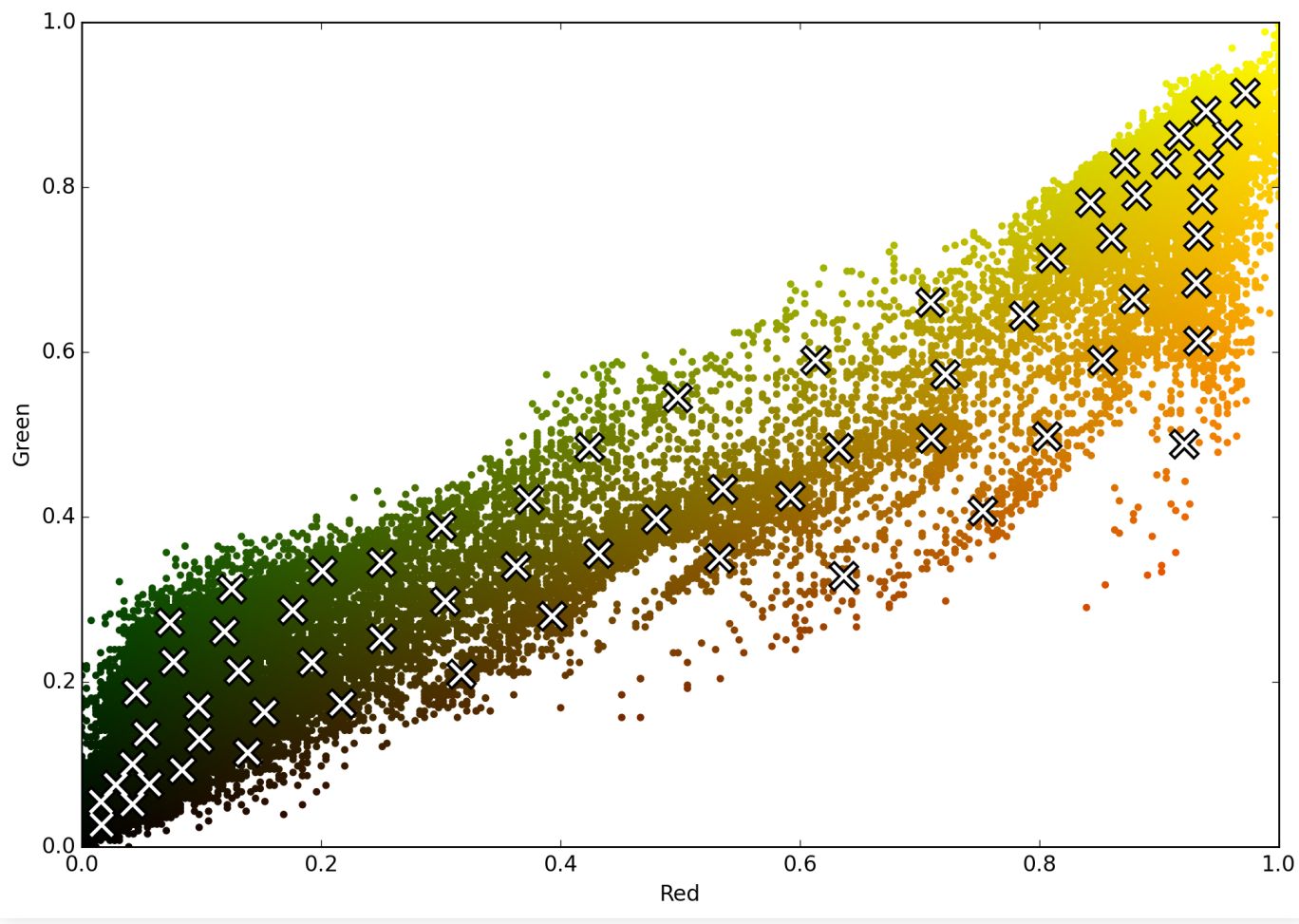
K-means color quantization

- Plot of all pixels green vs red



K-means color quantization

- Now we apply k-means with 64 centroids



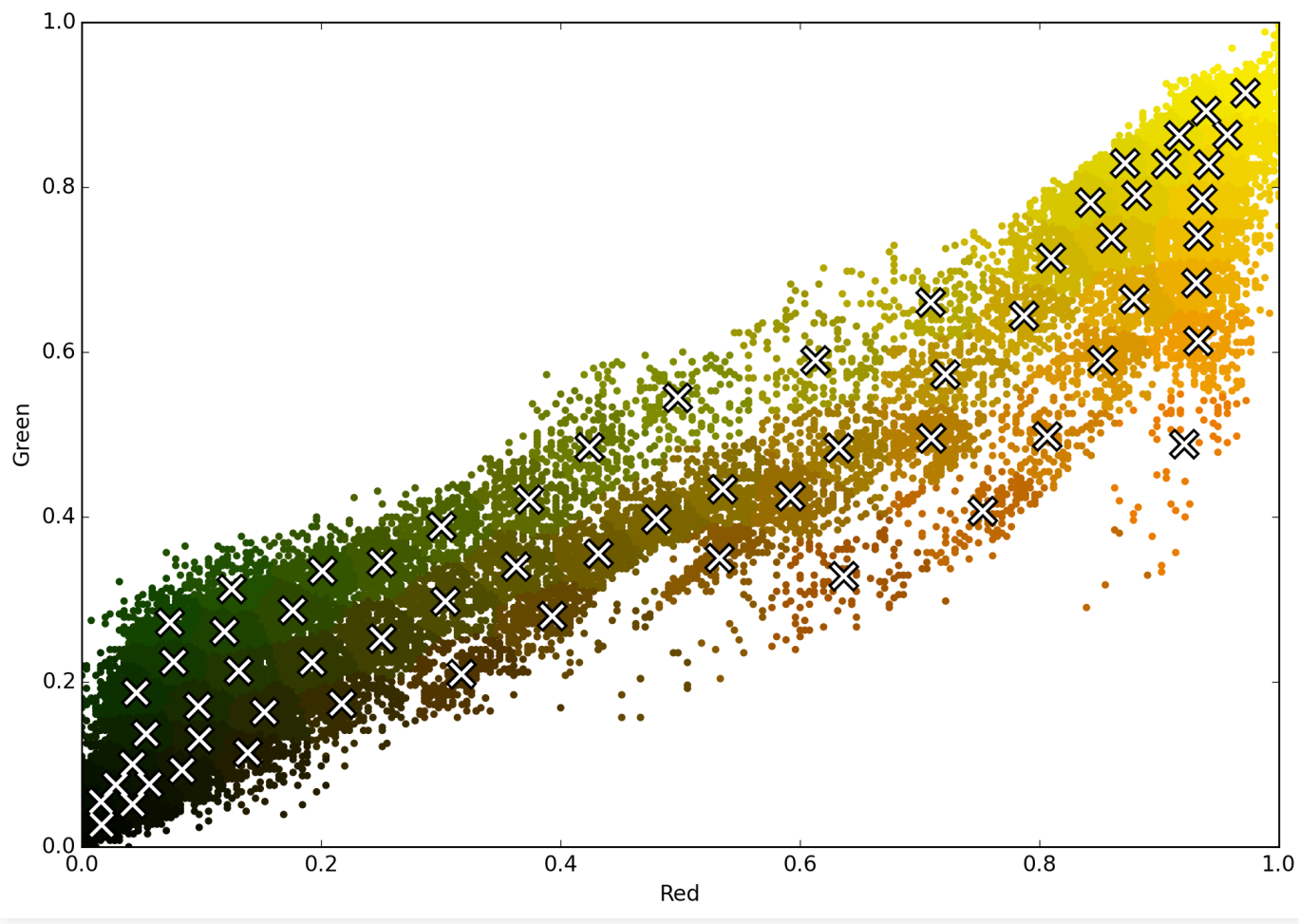
K-means color quantization

- Compute k-means, with 64 colors (6 bits)

```
n_colors = 64
kmeans = KMeans(n_clusters=n_colors).fit(image_array)
labels = kmeans.predict(image_array)
centroids = kmeans.cluster_centers_
plt.scatter(centroids[:, 0], centroids[:, 1], marker='x',
            color='k',s=200, linewidths=5)
plt.scatter(centroids[:, 0], centroids[:, 1], marker='x',
            color='w',s=150, linewidths=2)
plt.savefig('L17-rosa-plot-cs-'+str(n_colors)+'.png',
            dpi=200,bbox_inches='tight')
```

K-means color quantization

- And convert pixel colors to centroid



K-means color quantization

- Convert to centroid color and save

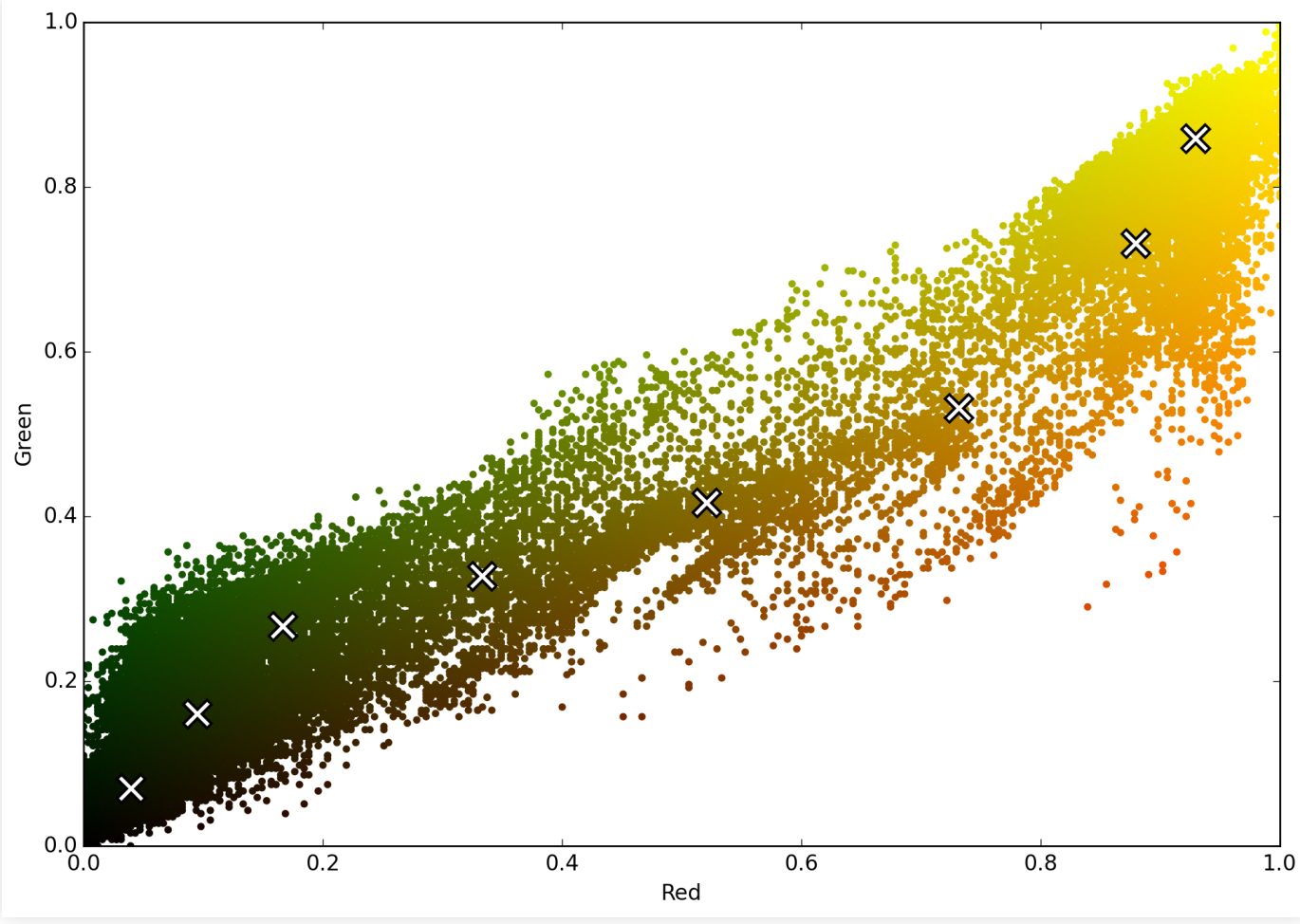
```
c_cols = np.zeros(cols.shape)
for ix in range(cols.shape[0]):
    c_cols[ix,:2]=centroids[labels[ix]]
# ...plot
plt.scatter(image_array[:, 0], image_array[:, 1], color=c_cols.tolist(), s=10)
# ...
imsave('rosa-comp'+str(n_colors)+'.png', np.reshape(c_cols,(w,h,3)))
```

K-means color quantization

- Compress from 24 bits to 6 bits

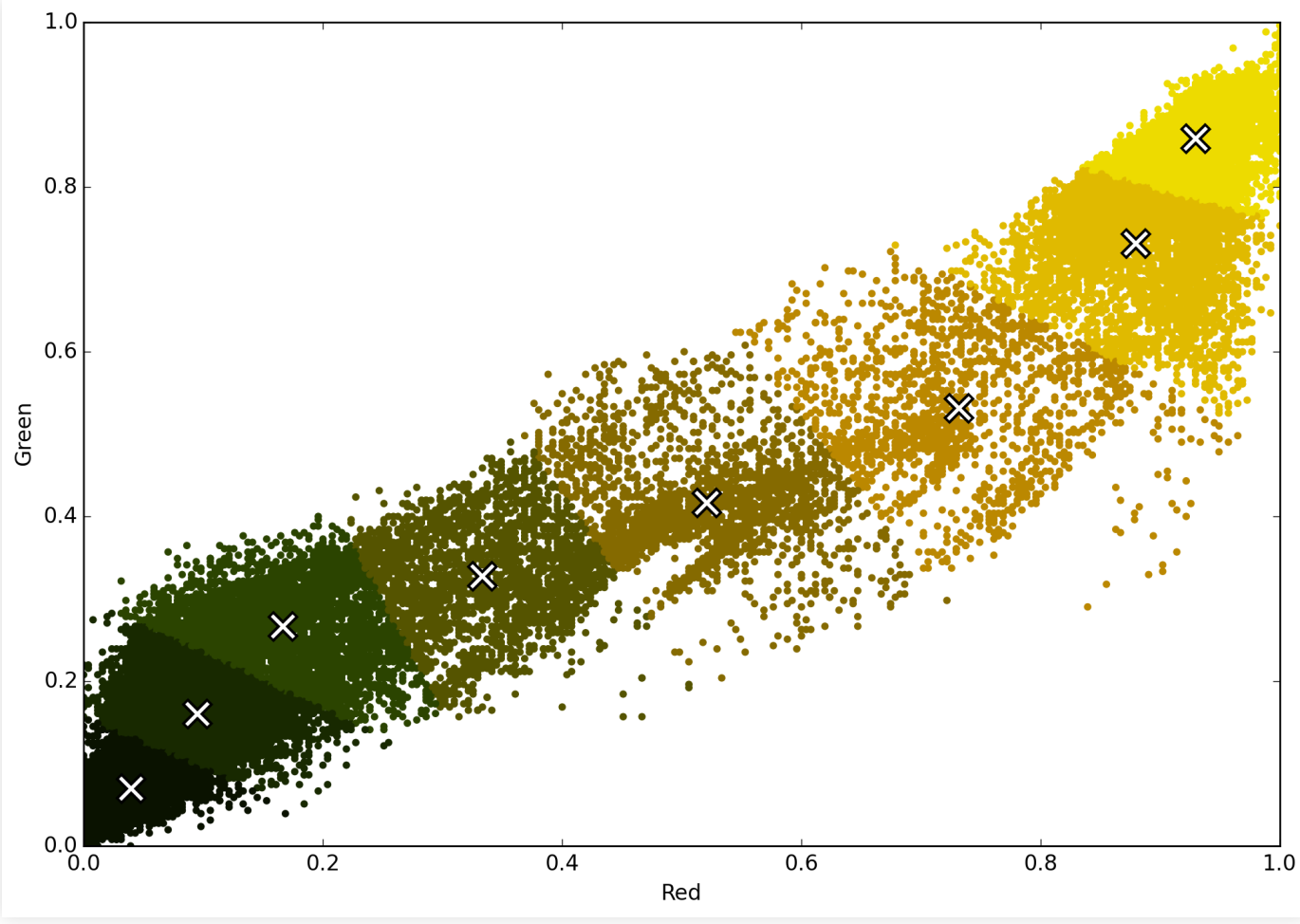
K-means color quantization

- Compress more, 8 centroids



K-means color quantization

- Convert pixel colors to centroid



K-means color quantization

- Compress from 24 bits to 3 bits

Summary

Summary

- Clustering: grouping data by similarity
- Hierarchical vs Partitional, Complete vs Partial
- Exclusive, Overlapping, Fuzzy or Probabilistic
- Types of clusters: Prototype, Contiguity, Density
- K-means: prototype, partitional
- EM algorithm: estimate, maximize, repeat

Further reading

- Bishop, Section 9.1 (and 9.3, intro, for EM)
- Alpaydin, Section 7.3
- Marsland, Chapter 9 (also relates k-means to SOM)